

程序员手册

南方信息企业有限公司

导　言

Amstrad cpc464 有一个设计良好速度快，功能强的Basic改写本，对于初学者，它可能相当复杂，对于习惯于其它BASIC 方言的人,可能会觉得难于适应 Amstrad，本书是为那些希望扩充自己在Amstad上设计BASIC程序的知识和技巧的人编写的,在解释每条 BASIC 命令的同时，本书详细描述了有用的程序，设计技术。这些技术使复杂的程序写起来很容易并可节省运行时间和存贮空间。

本书中有许多程序清单，其中一些是例子或从文本中摘录的重点文件，另一些则是标准结构，你可通过这些结构来编制你自己的程序。

第1～3章介绍了在Amstrad上进行程序设计的基本概念。经常使用的数组和程序设计技术在第 4 章中说明，此章的末尾，列出了一个简单的数据库，第 5 章说明如何用 Basic 命令建立游戏。第 6 章解释十进制，二进制和16进制数的关系，同明也说明布尔代数和信息如何通过位形译码。第 7 章介绍了机器码和汇编语言程序设计，并包括几个有用的子程序。第 8 、9 两章说明了Amstrad图形系统是如何操作的。第10章说明声音的产生，如何给乐曲标音并展示了Amstrad的一些独有的功能。第11章说明在盒式磁带机上记录程序，数据或屏幕图形。第12章讨论Basic的中断和提示方法，你可使用这些方法改进你的程序。

第一章　入　门

立即方式和延迟方式

你可以通过从键盘直接键入命令来指示你的 Amstrad 进行计算，显示字或划线等操作，这叫做“立即”或“直接”方式。你也可以键入一串这样的命令（一个程序）来让计算机执行，这叫做“程序”方式或“延迟”方式。你让计算机做的大部分操作将是执行程序，因为这个方法能使你恢复这些指令供你修改或者把它们存入磁带供以后再用，而用直接方式键入的命令则没有这些特点，更重要的是，程序能使计算机执行一系列复杂的操作，而直接方式的命令则不能。有一些命令不能用于直接方式，但它们对于检查程序的执行是很有用的，这一点我们以后将看到。直接方式的主要用处是键入和改变程序中的命令。

BASIC　词汇

你的 Amstrad 只认识有限的一组命令或指令。称为“Basic”的程序设计语言就是由这样的一组指令组成的。这些指令是一些英文词汇，如：LET、PRINT，LOCATE、MODE等，你可以用这些词汇组成一串指令（一个程序），这些指令存在计算机的存贮器中并能使计算机快速地执行复杂的操作。

有时，你也许会拼错一个词，例如：你可能把PRINT键入成PRNIT。因为计算机不认识这个词，则显示一个“出错信息”，在这种情况下显示的出错信息为“Synstax

error”。如果你要计算机执行不可能的操作，如：用零除一个数，计算机则产生相应的出错信息，这些将在以后的章节中讨论。

打印（PRINT）

无论在你的Amstrad中写入了什么程序，你都希望在屏幕上看到一些结果。你可能想显示数字，词或一些有用的符号。我们首先说明控制屏幕显示的最简单方法。

每当你键入一个命令后，你必须按标着ENTER的大蓝键，这就告诉计算机你想执行所键入的命令。我们用〈ENTER〉表示这个键。本书中的命令都是用大写字母表示，实际应用时，也可用小写字母，Amstrad 不区分大小写字母，。按标着CAPS的键，可对大小写字母进行转换。

显示字符（字母、数字、符号）的最简单方法是使用PRINT命令。例如：“PRINT 7〈ENTER〉”这个直接命令将在屏幕左上角显示出数字7。你可以把你的机器做为计算机使用：键入“PRINT 7 * 9〈ENTER〉”，则屏幕显示出63。注意，Basic用“*”代替乘，用“/”代替除。如果你键入“PRINT 18/6〈ENTER〉”，则在屏幕左边显示出3。在命令中，数字和符号间不必键入间隔，本书为清楚起见加入了间隔。但命令后必须留一间隔。

打印分隔符

你可能会注意到，当计算机执行完你的命令后，则在操作的结果的下一行显示“ok”。你是否希望7 * 9和18/6的结果在同一行显示呢？首先，你要知道，计算机在屏幕上打印了一些东西后则把光标移到下行的开始。但是通过使用分

（“!”）’可很容易得到这个码。为了重新定义一个字符，你必须发出两条指令。第一条指令是 SYMBOL AFTER n，它告诉计算机，你想把一个ASCII码大于几的字符重新定义。在我们的例子中，你可使用SYMBOL AFTER 31或SYMBOL AFTER32。把感叹号重新定义成我们上面说的字符，使用的命令是，‘SYMBOL33，255，193，161，145，137，133，131，255。第一个数33是被改变字符的ASCII码。后面的8个数是每行中建立的点的列值之和（图2.4）。你一旦重新定义了一个字符，这个字符便以新的面

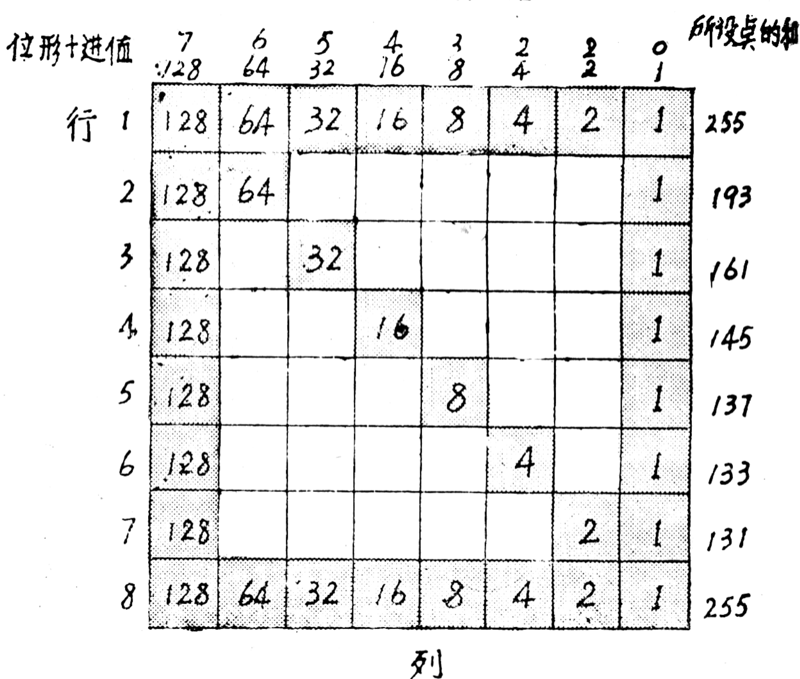


图2.4 为SYMBOL命令计算值

貌出现，而其它字符则不改变。为了使感叹号或任意一个重新定义的字符恢复原状，你必须发出一个SYMBOL AFTERn命令，这里的n是任意正整数。

数字变量和串变量

在本章的开头，我们讨论了变量。那些都是简单的变量，用于在计算机主存中存贮数字。通过程序，我们也可把字母、字甚至短句存入Amstrad。数字变量和字符或字符串变量之间唯一的区别就符串变量用$符号标识。在程序或直接命令中你可以使用‘LET A$ = “AMSTRADCPC 464”’这样的语句。命令PRINT A$将显示串变量A$所对应的一串字符。一个串变量最多可表示255个字符。没有赋与字符的串变量称为空的。

有一种方法可使串变量中包含数字，赋值语句‘LET-NUMPER$ = “7”是允许的，但必须认识到，这里的7和数字的7是十分不同的，这里的7仅仅是一个符号，而不是一个实际的数。

和数字变量一样，串变量可以有较长的名字并且名字不能以数字开头。你可以在数据库程序中使用象NAME $或ADDRESS$这样的串变量来存贮有关你朋友或用户的信息。你可以对数字变量和串变量使用同一名字，计算机可以区分它们，也就是说你可同时使用A和A$或NUMBER和NUMBER$做为变量名。

你不要混淆串变量和数字变量，否则将显示出错信息“Type mismatch”（类型不符）。象A = “HELLO”这样的赋值语句是非法的，因为“HELLO”是一个字符

串，而A＄一个数字变量，数字变量是只能包含数值 的。同样，WORD＄＝45也不允许。NAME＄＝VALUE 和PHONENO＝NOMBER＄也是错的，因为你不能 对数字变量和串变量交叉赋值。数字和字母是十分不同的数据类型，不能把二者混淆，虽然有办法使它们互相转换。

整数和整数变量

象前面描述的实数变量一样，Amstrad 有处理整数 的功能。此功能的优点是省存贮器且速度快。如果用百分比符号（％）把一个变量定义为整数变量，则可节省保存变量的存贮器，因为Amstrad可以不管小数以后的部分。由于存贮整数所需的空间少，存取整数所用的时间也就少。计算机可以区分变量A和A％，但为程序清楚起见，最好使用不同的变量名。

整数变量和数字变量的用法一样，但要记住必须使用整数符号％。也要记住，如果你把一个小数赋给整数变量，所存入的值将是不精确的。另一个限制是整数变量的赋值不能超过－32768至＋32768这个范围，否则，将显示出 错 信 息息“Overflow”（溢出）。

为了比较整数变量和实数变量的处理速度，试运行下面的程序：

```
1 REM Normal numerin variables
10 CONST =TIME
20 FOR COUNT =1 TO 1000
30 NEXT
40 CLS: DURATION =(TIME-CONST)/47
```

```
50 LOCATE 0, 10: PRINT"That took"; "DUR-
    ATION; "
senonds"
1  REM Integer numeric variables
10 CONST=TIME
20 FOR COUNT%=1 TO 1000
30 NEXT
40 CLS: DURATION=(TIME-CONST)/47
50 LOCATE 0, 10:PRINT "That took"; DURA
    TION; "seconds"
```

除了整数符号外，这两个程序是一样的。

定义变量（DEFSTR、DEFINT和DEFREAL）

这三个命令，允许你把一个单字母变量名用作串或整数。最好把这些命令放在程序的开头。例如：'DEFSTR A，表示把变量A定义为串变量。此命令可节省存贮空间（因为对于这个变量，你不需要每次都用$）。试一试下面的程序：

```
10 DEFSTR A
20 A= "This is a string"
30 PRINT A
40 DEFINT A
50 A=32000
60 PRINT A
70 DEFREAL A
80 A=1.2E^25
```

90 PRINT A

你可以用这些命令定义一组变量名。'DEFSTR A，G，W－Z '表示A，G及W与Z之间的字母都定义为串变量名。

你也许要花费一些时间来习惯于使用这些命令。对于初学者最好先不用它们。

第三章 字符串和键

输入（INPUT）

你可以在直接方式下键入命令，语句和程序。在程序运行期间，能从键盘输入也是有用的。这一点在游戏中是非常必要的，例如：玩游戏的人按一个键启动游戏开始。对于一个需要用户键入数据的数据库也是如此。

在程序期间，从键盘接收字符的命令是 INPUT。此命令允许输入数字变量或串变量，它使计算机在屏幕上显示一个问号并一直等到用户键入一串以〈ENTER〉结尾的字符，然后把这串字符赋与INPUT后边的变量。如果你的程序中有一个INPUT name $；则你键入的字符将赋给串变量name $。

因为数据类型是不相容的，当INPUT后边的变量是一个串变量时，用户键入一个数字是可以的，反之却不行。如果你的程序中有指令'INPUT number'，用户键入'seven〈ENTER〉'将使程序停止，并显示出错信息"Redo from start"（重新开始）。这个信息告诉用户：重新键入一个数字，这个错误的产生是由于字符串不能赋给一个数字变量，但是，把数字赋给串变量是可行的，并且这一点是很有用的。

INPUT本身没有多大作用，它只是显示一个问号，并不能告诉用户键入什么，因此需要在屏幕上显示一个提示。这样做最简单的方法是在INPUT前面使用一个 PRINT 语句。例如：

```
100 PPINT "Please enter your name";
110 INPUT name$
```

（以后我们将不使用 ENTER ，认为你已经记住了）。

注意，由于PRINT语句后面的分号，光标将留在提示行中。

通过扩展INPUT语句，你可得到同样的效果：

```
200 INPUT "Please enter your name"; name$
```

或

```
300 INPUT "Please enter your name", name$
```

在200行中，仍然会出现一个问号，但在300行中，逗号则把问号删除了。

IF，THEN和GOTO

IF、THEN和 GOTO 命令允许你对变量等项进行比较，并且控制程序执行的顺序。

通常，计算机从第一行开始处理程序，按顺序处理到最后一行。在第二章说明的FOR…NEXT允许循环对一些行重复执行多次，这种"改方向"是非常有用的。

假定，你希望程序用一个口令，以便不知道口令的人不能使用这个程序。口令可以是数字、字或短语。为了简单起见，我们用"secret"做口令。你程序的前几行应该是这样的：

```
10 CLS: INPUT "Please enter the password",
     password$
20 IF password$ < > "secret" THEN GOTO 10
```

30 PRINT “OK, let's get on with the rest of the program” 10行清屏和打印引号中的提示。然后计算机等待，直到用户键入若干个字符。并以〈ENTER〉结尾20行进行判别，如果键入的不是“secret”，则使计算机跳回10行，重新显示提示并等候输入，直到你键入了正确的口令为止。然后计算机执行30行和程序的以后部分，符号〈〉表示“不等”或“不同”，是用于比较变量的。

IF语句的通常形式是：IF（条件）THEN（操作）。条件部分比较变量，看其是否相同。操作部分可以是一个GOTO命令，PRINT语句或其它操作。在THEN后面，也可以有多个语句，只要条件成立，这些语句都将执行。例如：

```
1000 IF reply $ =answer $ THEN score=score+
10:LOCATE
 0, 20:PRINT "score so for"; score; :GOTO
500
```

当你读本书的时候，你将遇到许多IF…THEN语句的例子并且你会很快熟悉它的用法。

ELSE

ELSE可以在IF…THEN语句后面使用，意思为“否则”。它必须写在与其相应的IF语句的同一行。它指示当IF语句不成立时，将做什么。例如：

```
100 IF reply $ =answer $ THEN PRINT "Corr-
ect" ELSE PRINT "wrong"
```

在ELSE后面也可以有多个语句，例如：

750 IF reply $=answer $ THEN PRINT "Correct" :score=score+10 ELSE PRINT "wrong"; : score=score-5

在一个IF…THEN语句中使用另一个或多个IF…THEN语句，可以组成复杂的条件语句。例如：

830 IF a>b THEN q=q+1:a=b ELSE IF a<b THEN q=q-1:b=a

INKEY $

有时你不需要用户键入整个字符串，你只希望按一个键。一个典型的例子是：在游戏结束时，程序需要问玩游戏的人是否想再玩一次。为此机器显示"Do you want another go? Press Yfor yes, N for no"或简略地显示"Another go? …Y/N"。

你可以用INPUT语句来显示，但是，你每做一次选择必须按一次〈ENTER〉是很麻烦的。利用INKEY $我们可以只按一个键。INKEY $一个有用的方面是：这个功能所识别的字符不在屏幕上显示。

由于Amstrad可以很快地测试键盘，我们必须把INKEY $和IF…THEN及GOTO语句结合使用，以便可以等待一个键被按下。INKEY $仅测试一次键盘，因此下面的程序是行不通的：

```
100 PRINT "press y for yes, N for no"
110 response $=INKEY $
120 IF response $= "Y" THEN GOTO 10
130 IF response $ "N" THEN END
```

在这个例子中，除非用户在机器执行110行时按了一个“Y”或“N”，否则，程序将顺序执行下去。解决的方法是：要使机器重复执行110行，直到用户按一个键。为此，我们增加一行：115 IF response $ = “ ” THEN GOTO110′，这表示，如果用户没有按键，则转到110行。双引号“ ”表示空串。我们也必须处理这个事实：用户可能不按“Y”键或“N”键。

完整的程序如下：

```
100 PRINT "press y for yes, N for no"
110 response $ =INKEY $
115 IF response $ = "  " THEN GOTO 110
120 IF response $ = "Y" THEN GOTO 100
130 IF response $ = "N" THEN END
140 GOTO 110
```

同样，我们也可在115行中使用这样的语句：IF response $ < > “Y” AND response < > “N” THEN GOTO110。

串 处 理

Basic有许多处理字符组和串变量内容的内部功能。这些功能称为串处理，其中包括LEFT $，RIGHT $ MID $，LEN和INSTR。前三个产生字符串，后两个回送数字。

LEFT $

LEFT $用于拷贝字符串的左边，例如：‘PRINT LEFT（“Example”，3）’将显示字符串左边的三个字母‘Exa’。如果你对LEFT $键入了大于字符串长度的数

字，如‘PRINT LEFT $（“Example”，9）’，将显示整个字符串。在LEFT $中使用数字的唯一限制是不能用负数，使用小教则取整数部分，使用0则显示一个空串。上面的这些也适用于RIGHT $。

RIGHT $

这个功能拷贝字符串的右边，‘PRINT RIGHT $（“Second example”，5），将显示字母“ample”。

MID $

MID $比较复杂。它需要一个字符串和两个数字。第一个数字是字符从开头数的位置，第二个数字是要拷贝字符的拷量。例如：‘PRINT MID $（“Third example”，5，4）’将显示‘d ex ’。这是从第五个字符开始取的4个字符。同样，‘PRINTMID $（“Third example”，7，4）’将显示“exam”。你也可使用MID $从一个字符串中抽出一个字符。例如：‘PRINT MID $（“One character”，3，1）’将显示“e”。

MID $也可用于在字符串中插入字符。在Basic语言中，把它作为赋值语句是非常有用的，也是非常方便的。在这种用法中，它出现在等号的左边，并且把它表示的字符串加到等号右边的字符串中。例如：

```
10 a $ = "The first string"
20 PRINT a $
30 b $ = "final"
40 MID $( a $, 5 , 5 ) = b $
50 PRINT a $
```

UPPER $和LOWER $

这两个命令转换字符串的大小写。，PRINT UPPER $（“Amstrad”），将显示“AMSTRAD”；'PRINT LOWER $（“Amstrad”），将显示“amstrad”。

STRING $

STRING $产生一串同样的字符，它需要两个参数：所要重复的字符次数和ASCII码值或对应的串变量。它最普通的用处是产生一个花边，例如：

```
10 MODE 0
20 PRINT STRING $(20, "A")
30 LOCATE 0, 25:PRINT STRING $(19,
   67);
40 man $ = CHR $(249)
50 LOCATE 0, 2:PRINT STRING $(15,
man $)
```

注意：在20行中STRING $所给的字符在引号中，在30行中，所给出的是ASCII码，而在50行中所给出的是串变量。在这方面，STRING $是很灵活的。

LEN

LEN送回字符串的长度。'PRINT LEN（“A strin）is a group of characters”）'将显示数字33。（记住：间隔也是字符）。因为一个字符串不能超过255个字符并且一个空字符串不包含字符，LEN回送的总是0—255间的整数。

INSTR

INSTR是一个特殊的命令，它用于在一个字符串中查找另一个字符串，它需要三个参数：开始的字符位置，串本身

和要查找的字符。‘PRINT INSTR(1，“Amstrad”，“tra”）’将显示数字4，因为“tra”在“Amstrad”中的开始位置是4。省略第一参数，则计算机认为开始位置是1，因此上面的例子和‘PRINT INSTR(“Amstrad.”“tra”）’是一样的。‘PRINT INSTR(5，“Amstrad”,“tra”)，将送回0，因为“tra”是在第五个字符以前。你可以使用INSTR来检查是否用户已键入了一个包含给定词的句子。例如：

```
980 IF INSTR(reply $, " please" ) = 0
THEN PRINT
 "You 'll have to be more polite"
```

例子

上面讲的所有功能，可以把串变量转变为参数——你不必定义引号中实际的字符串，虽然，我们已用PRINT来显示这些功能的操作，但你也可把适用于一个串功能的结果赋给一个串变量。下面两个例子证明了这几点。

```
10 LET surname $ = "THATCHER"
20 PRINT LEFT $( surname $, 4 )
30 PRINT RIGHT $( surname $, 3 )
40 PRINT MID $( surname $, 2, 3 )
```

下面是一个短程序，需要用户键入自己的全名，姓和名用空格分开，然后把输入字符串分成两个名子。

```
10 CLS : MODE 2
20 LOCATE 0, 10 : INPUT "Please type your
   full name, first and last, then ENTER",
   full.name $
```

```
30 length=LEN(full.name $)
40 space.pos=INSTR(full.name $, "    ")
50 IF space.pos=0 THEN GOTO 10
60 first.name $=LEFT $(full.name $, space.
   pos-1 )
70 last.name $=RIGHT $(full.name $ $, len-
   gth-space.pos )
80 CLS
90 LOCATE 0, 10:PRINT "Thank you";first
   name $
```

把INPUT、FOR…NEXT STEP、MID $和LEN结合起来用，我们能使计算机执行象颠倒人名这样的功能。为此，我们先清屏，然后让用户键入自己的名字，并且必须提示用户这样做。我们可用INPUT来产生提示，并且把键入的字符存在串变量name $中。为此，可使用下面的程序：

```
10 MODE 2
20 INPUT "Please type in your first name,
   then press ENTER" name $
```

注意：上语句中的逗号可使引号不显示。

下面，我们需要从变量名为name $的字符串的最后一个字母开始循环，你最好使用FOR…NEXT循环语句，用一个负数的STER值向后循环，即从名字的右边到左边。我们需要知道字符串的长度以知道最后一个字符的位置，这一点可通过LEN做到。下面是整个程序：

```
10 MODE 1
20 INPUT "Please type your first name, then
```

```
   press ENTER", name $
30 FOR letter=LEN(name $)TO 1 STEP-1
40 PRINT MID $(name $, letter, 1);
50 NEXT letter
```

注意：40行的分号可使字母在同一行打印，并且长变量名可使程序易懂。

连接

连接表示把字符串用加号连在一起，例如：

```
10 LET surname $= "Jones"
20 LET first name $ = "David"
30 LET first. name $=first. name $t+ " " su-
   rname $
```

在10行和20行定义的两个字符串在30行被连接在一起并赋给第三个串变量。注意：中间加入的空格是为了把姓 和 名 分开。字符串不能相减，但你把MID $，LEFT $和RIGHT $联合使用可做到这一点。

SPACE $

SPACE $按给定的长度产生一串空格。例如：'alongblak $=SPACES(250)'。注意：括号中的字不能超过255。一个非文件功能SPC可以同PRINT一起使用，但是括号中的数字是以 40 为 模 的。因此， ' PRINT SPC(89)'将产生9个空格。因为SPC只能和PRINT一起使用，你不能象SPACES一样使用它。'LET along $=SPC(250)'将显示一个出错信息 "syntax error"(句 法错)。

数据类型转换

STR $和VAL这两个功能可以把字符串和数字互相 转换。

STR $

STR $用于把数字转变成对应的字符串。数字不能直接赋给串变量，‘LETa $ = 7 ’是一个非法命令并且产生 出错信息。但是STR $可以把数字连同正数前面的空格 和 负数前面的减号一起赋给一个串变量。例如：‘LETa $ =STR $ (7) ’。括号中的参数可以是数字变量，例如：‘LET value =19 : namber $ =STR $(value) ’。

VAL

VAL回送一个字符串的值，它检查字符串或串变量的数字内容并产生一个数字。例如：‘PRINT VAL (“1 2 3)回送数字123。‘PRINT VAL(“12A3”)， 回 送 12。如果字符串以符号 &开头，VAL则把字符串的其 余 部分算成16进制数。‘PPINT VAL(“&A)”，和 ‘PRINT VAL(“&” + chr $(65) ’都将产生数 字 10，因为字母A在16进制中表示为10。

特别有用的是，VAL可把通过INKEY $得到的单字母键入转换成数字，例如：

```
10 MODE 1
20 LOCATE  0 , 3 : PRINT “Please press a nu-
   mber between I and ൦”
30 akey $ =INKEY : IF akey $ = “   ” THEN30
40 number=VAL( akey $  )
```

```
50 IF number = 0 THEN 10
60 LOCATE 0, 7
70 PPNIT "That number squared is"; number
   *number
```

在这个例子中，数的平方通过数的自乘来计算。另一个方法是使用乘方符号：上箭头和数字2。例如：'square = number ↑ 2。这个符号也可用于立方等。

一些数字处理操作

RND

在许多程序中，使计算机产生一个随机数是非常有用的。Amstrad有这样的功能，称为RND。它回送0和1之间的随机数，因此如果你想得到较大的数，必须对结果做乘法。例如：'LET random = RND(1) *10'。这将产生0到9.99999999之间的数。为了把这些数转变为整数，你可以使用INT，ROUND，或FIX。

NT

INT把一个带有小数的数转变为整数。'PRINT INT(3.3)和'PRINT INT(3.6)都产生3。

FIX

FIX处理正数和INT一样，但处理负数得出的值大于用INT处理的负数值。

ROUND

ROUND对于数字的小数部分进行四舍五入，'PRINT ROUND(3.3)'将产生了，'PRINT ROUND(3.6)'将产生于4。

为了得到1至6之间的随机数，你可以使用下列程序：

```
100 LET random. number = RND(1) * 5
110 LET random. number = random. number + 1
120 LET random. number = ROUND (random.
    number)
```

100行把0和4.99999999之间的值赋给变量random. number'. 110行把这个值加1，使其成为1至5.99999999之间的值。120行把这些值取为1至6之间的数。这三行可以缩为一行：

```
100 random. number = ROUND((RND(1)*)
    + 1)
```

如果使用INT，则为：

```
100 random. number = INT(RND((1) * 6) + 1)
```

REM语句

REM表示注释，它在程序中使用可使程序易懂。它可用在程序的任何地方，计算机并不执行它，例如：

```
10 REM sample program
20 REM
30 REM Initialise variables
40 score = 0 : max. goes = 10
50 REM max. goes is the highest number of
   'turns' allowed
60 players = 5 : title $ = "HANGMAN" : REM
   title $ is the name of the game
```

注意：REM可用于分开两行（20行），也可以跟一些空格

（30行）并且可以用在冒号后面。REM也可以简写为单引号，例如：

```
100' Calculate average
110' Result will be average
120 IF number=0  THEN  average=0 :GOTO
    170: 'If number is zero, need to avoid
130' calculation or we'll get a
140' division by zero error
150 average=total/number
160 REM Calculation complete
170 IF average=0 THEN PRINT "Error"
180 REM Rest of program
```

当你刚开始程序设计时，应该用REM来帮你做标记，以易于发现程序中的问题。

ZONE

ZONE建立一个制表段来计算在PRINT语句中逗号后面的什么地方显示项目。ZONE只能给出1至255之间的数，并且把一个实数四舍五入。ZONE经常设到13。比较'PRINT "a"， "a"， "a"'和'ZONE 7 :PRINT "a"， "a"， "a"'。ZONE和PRINT USING连合使用可较容易地建立表格。

WIDTH

WIDTH用于设置打印机的行宽。这表示，Amstrad送给打印机一个字符的数量，还要送一个回车／换行（CR／LF)信号，以便打印下一行。WIDTH255停止自动CR／

LF，而让软件或打印机决定什么时候打印头回到左边和走一行纸。许多打印机有一个DIP开关来控制是送给打印机一个CR／LF还是由打印机自己决定（此时通常每行为80字符）。

如果你的Amstrad/打印机系统有产生一行空行的问题，可以试一下切断打印机电缆的14线。线14传送AUTO FEED XT信号。一个较严重的问题是：Amstrad仅送一个字节的后7位，因此ASCII128至255不能做为控制码送给打印机。如果图形字符不能象你希望的那样打出来，可能就是这个原因。

例子

这里是一个简单的程序，它说明了一些在本章中描述的原理，并说明如何使用这些功能。

```
10 MODE 1 :LOCATE 15, 0 :PRINT "GUE-
SSING GAME" :tries=0
20 anumber=ROUND((RND(1)*8)+1
25 REM Select a number between 1 and 9
30 LOCATE 0, 5 :PRINT "I've thought of a
   number between 1 and 9"
40 LOCATE 0,7 :PRINT "Press a number to
   guess it"
50 guess$=INKEY$:IF guess$= "" THEN
   GOTO 50
55 'Trap null $ in GUESS$
60 guess=VAL(guess$) :IF guess=0 THEN
   GOTO 50
```

```
65  'Convert $ to number with-VAL, if zero
    go back to line 50-get another key press
70 tries=tries+1 : REM Update nember of
    attempts
80 IF guess=anumcer THEN GOTO 110 : 'cor-
    rect
90 IF guess < anumber THEN LOCATE 0, 10 :
    PRINT guess; "is too low" :GOTO 50
100 IF guess > anumber THEN LOCATE 0, 10:
    PRINT guess; " is too High" :GOTO 50
105 REM
107  'Routine for a correct answer
108  '
110 CLS : LOCATE 0, 10 : PRINT guess; "is
    right"
120 LOCATE 3, 12 : PRINT "you got it in";
    tries; "tries"
130 LOCATE 10, 15 : PRINT "Another go…y/
    n"
140 akey $=UPPER $( INKEY $):IF akey $=
    "  " THEN 140
150  'Collect upper case character & convert
    to caps
160 IF akey $="Y" THEN 10
170 IF akey $="N" THEN CLS : END
180 GOTO 140:'akey $ neither Y or N,so rep-
```

eat line 140 until it is

10行设置屏幕方式并从0行15列开始显示程序的题目同时也把tries的初值置为0。20行把1至9之间的随机数赋给变量anumber。30行和40行显示提示信息。50行重复地测试键盘，直到有一个键被按，然后把键入的字符赋给串变量gues-$。60行测试guess $的值，如果为0，则重新执行50行。70行更新tries的值。80行检查guess值，如果正确则转到110行。90行检查guess值是否太小，如果是这样，便显示出相应的信息并转到50行来接到下一个guess。100行测试guess值如果大于随机数则显示相应的信息并转入50行。110行以后的行处理正确的答案，并使用Y/N来测试，再使UPPER $把键入的字母变为大写。

你也将注意到，我们没使用LET并且在THEN后面没有使用GOTO，即'IF（条件）THEN GOTO 50'和'IF（条件）THEN 50'是一样的。不要把NEXT放在IF…THEN的行中，因为如果IF后面的条件不成立，则不执行INEXT。

下面是一个用INKEY $来代替INPUT的程序。INPUT有许多优点，但当键入一个长字符串或数字时，用户可造成错误。商业软件经常显示一个带括号的提示，如：'Please enter a 3-digit number〈 〉'。光标被置于左括号后的空格处，用户可在括号之间移动光标，但不能超出。这一点Amstrad是易于做到的。由于使用了INKEY $，你可使之适于你的需要。例如，你可以使用一个标志，当调用这样的程序时，不接收非数字或非字母字符，你也可重新定义光标字符、及输入长度等。计算机不接收小于32和大于122的ASC

II码。

为了使用这个程序，你必须把'prompt $'定义为显示信息，把'maxlen'定义为键入字符的最大数，把'row'和'col'定为定提示或输入显示的开始行和列。当用户按〈ENTER〉时，子程序在输入给串变量'enter$"任何字符的同时返回主程序。

```
10 MODE 1
20 col=1 :row=10:maxlen=5
40 prompt $="Enter a 5 digit number"
50 GOSUB 10000
60 MODE 1
70 PRINT"You entered"; entry $
80 END
10000 akey $="  ":entry $= "  "
10010 enter $=CHR $(13):leftarrow=242
10020 curschar=95
10030 cursor $=CHR $(32)+CHR $(8)+CHR
      $(curschar)+CHR $(8)
10040 blank $=STRING $ (maxlen+1,8)
10050 LOCATE col, row
10060 PRINT prompt $;"<"; SPACE $ (max
    len);">";
10070 PRINT blank $; cursor $;
10080 akey $ =INKEY $
10090 IF akey $ ="  " THEN 10080
10100 akey=ASC(akey $ )
```

```
10110 entrylen=LEN(entry$)
10120 IF akey=leftarrow AND entrylen>0
      THEN GOSUB 10200:GOTO 10080
10130 IF akey=leftarrow AND entrylen<1
      THEN 10080
10140 IF akey$=enter$ AND LEN(entry$)
      =maxlen THEN RETURN
10150 IF(akey<32 OR akey>122)THEN 10080
10160 entry$=entry$+akey$
10170 entrylen=entrylen+1
10180 IF entrylen>maxlen THEN entry$=LE
      FT$(entry$,maxlen):GOTO 10080
10190 PRINT akey$
10200 IF entrylen<maxlen THEN PRINT curs
      or$;
10210 GOTO 10080
10220 entry$=LEFT$(entry$, entrylen-1)
10230 PRINT CHR$(8);cursor$;
10240 RETURN
```

键（keys）

KEY

Amstrad键盘是“软”的，你可以改变任何键产生的字符。这就是所谓的冗余特性。这个特性对程序员是很有用的。例如：当编制一个使用MODE 2 的程序时，每隔几分钟就键入'MODE 1 ：LIST'是很麻烦的，你可用KEY命令把

命令串赋给任意一个数字键。下面的例子把引号中的字符串赋给数字键 7 ：

KEY 7 ，"MODE 1 LIST" + CHR $（13）

注意：赋给一个键的指令必须在引号中，在这字符串的后面必须用加号连接一个CHR $（13），以便得到一个自动回车，所使用的键号是在Amstrad手册附录 3 中给出的码。在这个例子中你可以看到Amstrad自动地把128加到了所提供的数上。'KEY 135,"MODE 1 ：LIST" + CHR $（13）'与上例是一样的。

你只能用128至159间的31个键来扩展字符串。注意，所有小键盘上的键（除了小ENTER键外）都产生同样的值，如果SHIFT或CTRL也被按下的话，小ENTER和SIHFT，一起通常产生139，如果CTRL 如果CTRL也被按则产生1 0。当Amtrad开通时，如果和CTRL同时按这个键被设为产生RUN"〈ENTER〉。有120个字节分配给所有的KEY用作赋值。

KEY DEF

你可用KEY DEF来改变一个键所产生的ASCII码。例如：为了使大ENTER键产生大写的字母 A，你可以用'KEY DEF18，0，65，65，65'，第一个数对应这个键，第二个数表示这个键是否连发。0 表示不连发，1 表示连发，其它数则产生出错信息"lmproper argument"。后面的三个数是这个键将产生的ASCII码。其中第一个是正常字符，第二个是换挡的字符，最后一个II同时按CTRL键所产生的字符。如果你只想改变其中的一个，只需不写你不想改变的值。例如：你想改变 2 键的CTRL值，你可用'KEY

DEF71…32′。这使CTRL-2产生一个空格（ASCII码为32），但不影响此键的连发、正常或换挡字符的产生。

SPEED KEY

大多数Amstrad键可自动连发(即当按下一个键时，产生一小会儿延迟，然后以规则的间隔连发这个字符。使用SPEED KEY你可以改变延迟时间和连发间隔。如果你有大量的拷贝编辑要做或者使游戏快速地响应键盘，则加快重复速度是很有用的。

SPEED KEY需要2个参数：开始连发前的延迟和连发的间隔。两个参数都以1/50秒为单位。'SPEED KEY 100，50′表示键按下2秒后开始重复，重复的间隔为1秒。如果你在一个程序中设置了很短的延迟和很快的重复如'SPEED KEY 1,1′，你必须小心，因为当程序结束时，你想中断它，则会发现键盘失效。因为键将被按下瞬间连发，而且连发得很快，每按一次键可连发6个字符。你要在程序末尾把这两个值恢复为默认值或你所要求的值。恢复默认值的快速方法是使用"CALL &BBOC"，并且你应记住捕获错误和中断，转到程序的结尾，例如：

```
10 ON ERROR GOTO 10000
20 ON BREAK GOSUB 10000
30 REM Program
40  '
999 GOTO 10000：REM END
9999  'Reset key delay and repeat
10000 CALL &BB00
```

注意：'CALL &BB00′把所有的键都设成默认的ASCII

码。ROM程序重新设置键盘管理系统，其中包括赋值，连发速度等等。

禁止ENTER键

使用KEY DEF可以在程序运行时至少进行一次保护。例如：'KEY DEF18，0，0，0，0'将使大ENTER键什么也不产生。同样，'KEY 139。" "'将使小ENTER键不起作用。为了完成这个保护，你还必须使用'KEY DEF 38，1，109，77，0，这使CTRL-M也不产生回车动作（109和77分别是m的小写和大写ASCII码）。此时，用户想中断一个程序时，则是不可能的，因为所有的ENTER键都不起作用，象使用其它保护技术一样，你必须确保你的程序在使用这些命令以前执行完，你也必须重写输入程序来适应ENTER的一些其它改变，从而使程序可以接收用户输入。

虽然有32个可用的扩展键（128—159），但手册中只说明140个键，那么141—159号键在什么地方呢？答案是：你必须自己为这些键赋值。例如'KEY DEF 38，1，109，77，155'把155赋给m键。然后，你可以使用象'KEY 155，"RENUN"+CHR $（13）'这样的命令把一个功能赋给CTRL-M。

INKEY

INKEY允许你检查一个给定键的状态，它非常象INKEY $。例如：

100 IF INKEY（38）=0 THEN GOTO 1000

括号中的数是检查的键号，用法和KEY DEF是一样。根据键被按，或没按或同SHIFT一起按还是同CTRL一起按，INKEY回送四个值之一，这些值在下表给出：

INKEY值	表示状态
-1	没按键
0	按键
32	按键+SHIFT
128	按键+CTRL
160	按键+SHIET+CTRL

由于使用INKEY可以设一个键是不是被按和SHIFT和CLRT 是否也被按，所以INKEY可以代替 INKEY $。例如，在43号键(Y)被按的条件下，你想让程序转到5000，你可使用IF 'INKEY(43) < > -1 THEN GOTO 5000′，INKEY也允许你检查是否组合键SHIFT或CTRL被按。而处理这类事情使用INKEY $键是很麻烦的。INKEY 和 KEY DEF一起使用可允许你使用经过程序处理的复杂键盘。

第四章 子程序、数组和系统功能

子程序

子程序是可以从程序的任何部分调用的一段程序编码。使用子程序可节省程序空间。例如：你有一个在不同点停止的程序并要求用户按空格来继续执行。这个操作最好用子程序。这个程序的主要部分如下：

```
1000 LOCATE 25, 1:PRINT "please press the
    space bar to continue"
1010 A$=INKEY$:IF a$= "  " THEN1010
1020 IF A$<>CHRS$(32)THEN1010
1030  'On with the program
```

这段程序在屏幕底行显示一个提示，然后在100行和 1020 行之间循环，直到用户按了间隔键。如果你在程序中多次用这段程序从而多次地写它是很麻烦的。你可以把这段程序变为子程序，这需要在1030行加上 RETURN。然后，当你想执行这个操作时，必须使用命令GOSUB1000。

GOSUB告诉计算机跳到给定的行号，并且从那里开始执行。当执行到REUTRN命令时，则跳到GOTO命令的下一行。为了看看GOSUB如何工作，键入并运行下面的程序：

```
10 CLS
20 GOSUB 1000 : REM space bar subroutine
30 CLS
40 PRINT "Once more please"
50 GOSUB 1000 : REM space bar subroutine
60 CLS
70 LOCATE 10, 10
80 PRINT "THE END"
90 END
```

在你的整个程序中使用这样的自含子程序是很有用的。它们可较容易地产生表目，但用大量的REM来编制表目时要非常仔细。

使用FOR…NEXT循环可以把操作重复若干次，但也有同样效果的其它方法。一个方法是和GOTO一起使用IF…THEN。下面两个程序有同样的效果。

```
10 FOR COUNT=10 TO 100 STEP 10
20 PRINT"The square root
   of"; COUNT; "is"; SQR(COUNT)
30 NEXT
```

```
10 COUNT=10
20 PRINT "The square root
   of"; COUNT; " is"; SQR(COUNT)
30 COUNT=COUNT+10
40 IF COUNT<=100 THEN GOTO 20
```

第二个例子较长一点，但是当你不能确定循环多少次或在循环期间想改变循环次数时，这个方法是很有用的。

WHILE和WEND

还有两个象IF…THEN这样在一起的命令可控制循环。它们是WHILE和WEND。上面的两段程序可用下面的例子完成：

```
10 COUNT =10
20 WHILE COUNT < =100
30 PRINT"The square root
   of"; COUNT; "is"; SQR(COUNT)
40 COUNT =COUNT+10
50 WEND
```

这样的控制指令比FOR…NEXT循环有许多优点。通常，我们不知道确切的循环次教，使用这两个命令，解决了这个未满足条件而遗留FON…NEXT循环的问题，我们考虑下面的程序：

```
10 FOR avalue = 1 TO 10000
20 akey $ = inkey $
30 IF akey $ = CHR $(32)THEN 50
40  NEXT
50 REM on with the program
```

这个程序在程序中产生一个暂停，直到用户按一个间隔键或循环10000次为止。如果用户在循环完成前按间隔键，则仍然有一些FOR…NEXT的循环次数未解决。如果这个现象太频繁，则会产生问题。

ON GOTO／GOSUB

ON GOTO和GOSUB把类似下面的操作：

```
1000 IF option = 1 THEN GOTO 2010
```

```
1010 IF option = 2  THEN GOTO 5350
1020 IF option = 3  THEN GOTO 6870
1030 IF option = 4  THEN GOTO 8000
```

变为：1000 ON option GOTO 2010, 5350, 6870, 8000

这使程序简化，因此在键入程序时不容易出错。

ON GOSUB的用法和ON GOTO一样。对于这两个命令来说，如果所选的变量（你也可用数字表达式）值为0，则系统不执行ON…语句，如果算出的值大于所列行号的数目时也是如此。

当遇到RETURN时，ONGOSUB不把控制转到下一行。控制转到调用这个子程序的语句的下一行。ONGOSUB是菜单选择中的特殊值。使用每个选项的第一个字母和INSTR及GOSUB，可使菜单的设计变得简单。见下例：

```
10 CLS
20 PRINT "Q =Quit"
30 PRINT "N =Next"
40 PRINT "P =Previous"
50 PRINT "D= Delete"
60 Print "A =Amend"
70 menu $ = "QNPDA"
80 akey $ =INKEY $ : IF akey $ =" "THEN  80
90 Choice = INSTR ( menu $, akey $ )
100 IF choice = 0  THEN 80
110 IF choice = 1  THEN CLS : END
```

```
120 ON choice GOSUB 2000, 3000, 4000, 5000
130 GOTO 10
```

功　能

DEF FN

使用DEF FN可建立用户定义的功能，这些功不能做象发声、划线、打印结果这样的事。它们经常用于数字或字符串操作。一个很普通的用法是在程序的任意点产生随机数，而不需要重复使用 'random number = INT（RND）（1）*（100）+1′这样的语句。使用DEF FN 你可以在操作程序开始建立这样的功能：10 DEF FN rand(number) = INT（RND(1) * number）+ 1 然后，每当需要1到number之间的随机数时，你可以调用这个功能，就象调用子程序：'100x = FN rand（100）′一样。这个语句把100送给功能'rand′，然后在表达式'INT（RND(1) * number) + 1′中来代替number的现行值，这就给变量'ranbnum′一个1到100之间的整数值。

功能不限于对括号中的数的操作。它可以使用你程序中的任何变量，但仅回送一项。例如，如果你想产生两个值之间的随机数，以便根据变量"高"和"低"来存贮：

```
10 DEF FN rand(number) = INT(RND(1) * (high
    —low ) + low
```

功能也可对字符串操作。如果一个功能是接收并且回送字符串，则必须在后面加 $或串标识符。下例回送数字的串表示，并且不带符号：

```
DEF FN strip $(number) =
```

MID $(STR $(number), 2, LEN(STR $(number))-1)

为了使用这个功能，你可以这样调用：

1000 number $=FN strip $(number)

如果在一个功能中发现一个错误，出错信息将指出调用这个功能的语句的行号，而不是功能定义本身的行号。

数　　组

数组处理是Basic最有用的功能之一。一旦你理解了这个概念并对建立数组很熟练，你将发现，在程序中使用数组将使程序设计较简单。

数组是存贮数据的结构，可以认为它是一个表，你可以有实数、整数和字符串数组。数组用DIM来定义，它为数组在内存建立存贮空间。

最简单的一类数组是一维整数数组。可以把它想象为一个数值表。当我们用'DIM array%(10)'定义一个整数数组时，Amstrad为数据存贮建立10个小空间，我们使用赋值语句"="可以把数据装在这些小空间中。因此'LET array%(1) =99'把99放入数组的第一个单元括号中的数称为下标。同样地，'array%(5)=ROUND (divisor/dividend)'把表达式的结果放入数组的第五个单元。我们可以象普通的变量一样存取数组中的数。'LET number = array% (7)'把数组第7单元的值赋给变量number。见图4.1。

数组是很有用的，因为它允许程序员很快地存取或修改数据。例如，我们可用数组存贮一个数的平方，以便需要时

不必重新计算：

```
10 DIM square%(30)
20 FOR number = 1  TO 30
30 square%(number) = number ↑ 2
40 NEXT
```

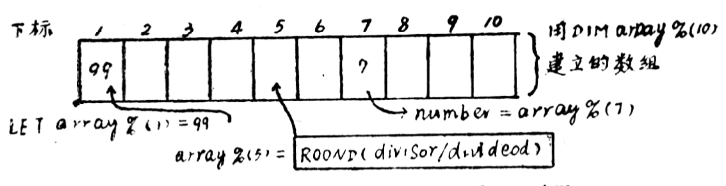


图4.1 数组'array%'的图示及一些操作的图解

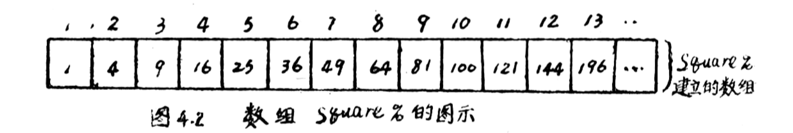


图4.2 数组 Square%的图示

这个数组的表（图4.2）称为查找表，把数值算好并存起来来后，存取时则比每次都重新计算快得多。查找表可以在程序开始时，从磁带文件中装入。

这个表说明了变量如何作为标号来用，变量 number 对应于所有的数组单元，因为number是FOR…NEXT结构的

循环计数，其值从1到30，因此可以作为标号。

整数数组可以处理-32768到+32768之间的整数。如果对整数组单元的赋值超出这个范围，则产生出错信息'Overflow'（溢出）。如果要把实数赋给整数组，则先要对实数使用'INT'命令。

实数组用于带小数点的数，如：

```
10 DIM realarray(100)
20 FOR eachcell = 1 TO 100
30 realarray(eachcell) = RND(1)
40 NEXT
```

实数组比整数组占用较多的存贮空间并且存取时所花的时间也较长，因此，只有必要时才用。字符串组也是可能的，在这种情况下，每次送的不是一个数而是一个字符串。每个字符串最多可达255个字符。例如：

```
10 DIM French $(20)
20 French $(1) = "un"
30 French $(2) = "deux"
40 French $(3) = "trois"
50 REM rest of definitions
1000 FOR count = 1 TO 20
1010 PRINT count; "="; French $ (count)
1020 NEXT
```

你可以不用DIM来使用小于10个单元的数组，如：

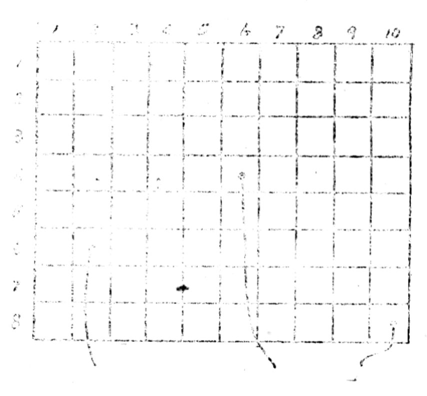
```
10 FOR cell = 1 TO 10
20 array(cell) = SQR(cell)
30 NEXT
```

注意：有 0 下标的数组对于存贮数据是很有用的，下面的子程序计算数组中数的总和，就是这样一个例子：

```
1000 total = 0
1010 FOR cell = 1  TO no. cells
1020 total =total+array(cell)
1030 NEXT
1040 array( 0 ) =total
1050 RETURN
```

多维数组

最经常使用的数组是多维数组，其中最简单的是两维数组。两维数组中的每项有两个下标，可以把它们想象为坐标纸上的行和列，（见图4.3）



用DIM array（8，10）建立的实数组。

数组（6,2）　　数组（4,6）　　数组（8,10）

图4.3　两维坐标数组的图示

在存贮器中建立这样的数组需要对DIM稍加扩展。例如，我们可以使用一个两维字符串组来在一个简单的数据库

中存贮姓名，地址和电话号码。这个数组可由语句 '10 DIM addressbook $ (100,3)' 来建立，从而为一个100 行 3 列 的字符串组开辟空间。我们把名子存在第一列，地址存在第二列，电话号码存在第三列。

为了把数据加入数组，可用下列程序：

```
200 LET addressbook $(1,1) =Fred Smith
210 LET addressbook $(1,2) =10, The Avenue
220 LET addressbook $ (1,3) =01—678—5478
```

图4.4将助你理解这个原理。

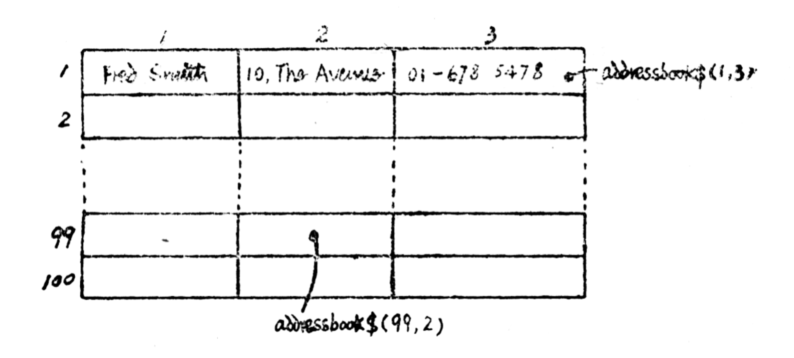


图4.4　字符串组 'addressbook $ 的图示

作为一个数组处理的例子，让我们来想象一个100 行20 列的数字组，编一个子程序来计算每列的和及平均值并把结果放入每列的最后两行（101行和102行）：

```
1000 FOR column =1 TO 100
1010 total = 0
1020 FOR row =1 TO 20
1030 total=total+array(row, column )
```

```
1040 NEXT row
1050 array(101, column ) =total
1060 average=total/100
1070 array(102)=average
1080 NEXT column
1090 RETURN
```

通过 'GOSUB 1000 '我们可以使用这个子程序经常修改数组的和及平均值，然后可以从数组的最后两行得到有关的值。（图4.5）。

在101行存贮的和

在102行存贮的平均值

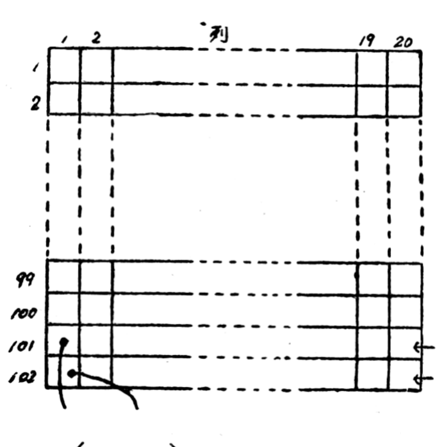


图4.5　利用数组的行存贮附加信息

也可以有多于2维的数组：'DIM large array$(20, 20, 20) '是非常合法的，但占用较多的存贮空间并且较难

建立它的概念。当需要一个三维数据时，我们可使用这样的数组，你可以把它想象为一个立体方块（图4.6）。在Basic程序中很难发现三维以上的数组，但Basic有处理这样数组的能力。

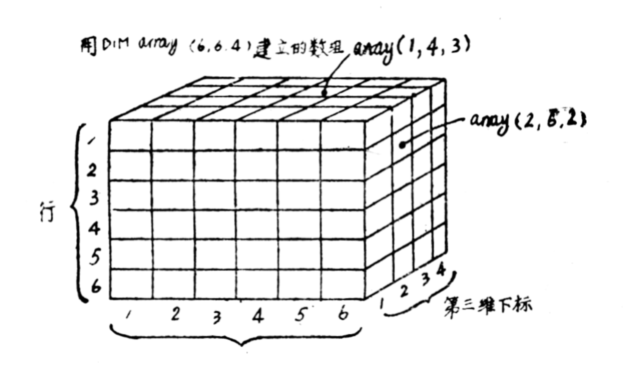


图4.6　3 维列数组的图示

动物游戏

初看起来好象很困难的程序设计问题，通常可用数组来简单地解决。一个这样的例子是动物游戏。在这个游戏中，你想出一个动物，然后计算机猜这个动物是什么。这个游戏是以这样的方法设计的：计算机学习新动物并问新问题。

在下面的例子中，计算机从只知道驼鹿和鲸鱼这两种动物开始，它们之间的区别是鲸鱼生活在水里。当计算机问：是否这个动物生活在水里？答案是Yes。计算机将问：是否你想的动物是鲸鱼。如果不是，计算机将问：这个动物是什么，并问这个动物和鲸鱼间的区别。然后重新开始。再让你

想一个动物。象这样简单的游戏，要把它编成程序是不容易的。这里提供了一个简单的程序，你可以修改它以适于你的需要。你可以把数组的初始内容改为植物，飞机或其它你感兴趣的东西。

这个程序使用一维数组存贮数据和问题。把信息存贮的方法想象为二元树是最容易的。（见图4.7）。问题是如何用一个一维数组表示这个两维结构。（是可以使用两维数组的，但会浪费大量存贮器。）数的每个“结点”存贮一个问题。

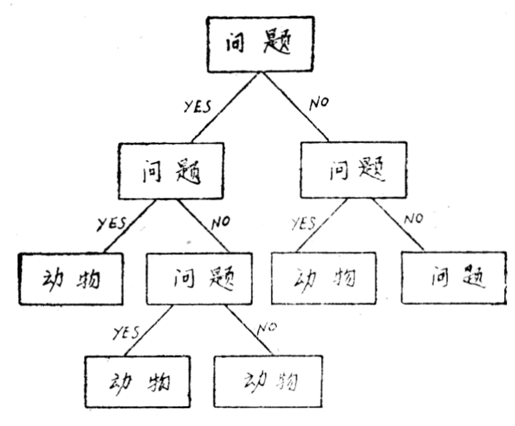


图4.7 动物游戏所用的二元树

或一个动物。动物名用符号“1”标明，当学习了新动物时，把其名字和相应的问题存入数组，并把其他项目做相应的修改，下面是这个程序：

```
10 DIM array$(1000) : array$(1) = "Does it li
   ve in water"
```

```
20 array$(2)="I moose":array$(3)="I whale"
30 CLS: element=1
40 this$=array$(element)
50 IF LEFT$(this$, 1) = "I" THEN GOS
   UB 80:GOTO 30
60 PRINT this$; "? " :GOSUB 180: IF ans-
   wer$ = "Y" THEN element=(2 * eleme-
   nt)+1 CLSE element=(element * 2)
70 GOTO 40
80 this$=RIGHT$(this$, LEN(this$)-1)
   :PHINT "Is it a" I this$; "? " :GOSUB
   180
90 IF answer$ = "Y" THEN PRINT "I gues-
   sed it-want another go? :GOSUB 180: IF
   answer$ = "Y" THEN 30 ELSE END
100 PRINT "I give up, what is it? "
110 INPUT it$
120 PRINT"Type in a question which
   will ":PRINT" distinguish between a ";
it$; " and a "; this$: PRINT" The question
   should start with, Does it, Is it etc. " :
   PRINT "The answer must be either yes or
   no"
130 INPUT query$
140 PRINT "what's the answer for a"; it$;
   "? " :GOSUB 180:it$ = "I" +it$:this$ =
```

```
   "I" +this$
150 IF answer$ = "Y" THEN array$ ( 2 *
   element ) = this$ :array$( (2 * element)
   +1 ) =it$ ELSE array$(2 * element)
   =it$:array*( (2 * element ) +1 ) =this$
160 array$ ( element ) =query$
170 RETURN
180 answer$ =LOWER$(INKEY$ ):IF ans w
   er$ = " " THEN      180
190 IF INSTR( "yn", answer$ ) = 0  THEN
   180 ELSE
   RETURN
```

系统功能

Amstrad有若干系统功能，它们对程序员是很有用 的。在本段中，我们将介绍其中的一些功能，其它功能将在后几章中说明。

ERASE

数组可以占有大量存贮器，有时，要收回这些存贮空间以供其它数组使用。ERASE命令可清除数组，把其所占的空间空出，以允许程序员使用。例如：

```
1000 ERASE real. array
1010 ERASE integer array%
1020 ERASE array$
```

同样，也可以这样使用：

```
1000 ERASE real. array, integer array%,
   array$
```

如果试图清除不存在的数组，则产生出错信息 'improper argument '(不正确的参数)。

READ和DATA

READ和DATA是Basic程序员的特殊工具。它们允许你在程序中存贮数据而不需使用多个象下列那样的赋值语句：

```
100 xcentre=320
110 ycentre=200
120 radius=10
```

我们可以使用下面的简单语句：

```
100 READ xcentre
110 READ ycetre
120 READ radius
130 DATA 320
140 DATA 200
150 DATA 10
```

我们也可使用更简单的形式：

```
100 READ xcentre, ycentre, radius
110 DATA 320, 200, 10
```

为了了解READ和DATA如何工作，想象一个数据指针。在程序的开始，这个指针指向第一个DATA语句的第一项。如果没有DATA语句，你发出一个READ，程序将停止，并显示出错信息 'DATA exhausted '(数据用

完）。如果你读的项目超出DATA语句中的项目，也显示这个出错信息。每当遇到READ语句时，数据指针所指的值或字符串将存入存贮器并赋给READ命令指出的变量。

READ可把字符串赋给串变量，例如：

```
100 READ name$, address$, phones$
500 DATA Fred Bloggs, 13 The Crescent Bury-
    Lancs, 0893—65734
```

注意：DATA语句可在程序的任何地方出现。

对于上例，你如何能把一个逗号放入DATA语句的字符串中，而使Amstrad不把这个逗号看成各项间的分隔符？答案是，可以把这样的字符串用引号括起来，例如：

```
100 READ name$, address$, phones$
5000 DATA Fred Bloggs, "13, The Crescent,
    Bury, Lancs.", 0893—65734
```

在一个DATA语句中，你可把字符串和数字混合用，但要注意，不要在数据指针指向一个字符串时使Amstrad读一个数字值，否则将出错。

READ和DATD对于填入数组是很有用的，在一个简单的数据库中，你可以定义这样一个数组'DIM data$（100，3）'来存贮磁带、书或邮票的信息。这个数组所需要的信息可放在DATA语句中，并在程序的开始读入数组。

让我们看一个简单的例子，这里我们想存贮音乐的标题、作者和种类，这个数组将有100行3列，并且我们想从DATA语句中读入信息。这个数组的装入程序可以这样开始：

```
10 DIM array$(100, 3)
20 row =1
30 READ array$(row, 1)
40 IF array$(row, 1) = "ZZZ" THEN 100
50 REM "ZZZ" signifies end of data
60 READ array$(row, 2), array$(row,3)
70 row=row+1
80 GOTO 30
100 REM all DATA read in
110 'Rest of program
1000 REM data in order title, artist, type
1010 DATA Beat, King Crimson, jazz
1020 DATA Holland, Beach Boys, Pop
1030 DATA Sketches of Spain, Miles Davies
     /Gil Evans, Jazz
1040 DATA The Four Seasons, Vivaldi, Class-
     ical
1050 DATA ZZZ
```

为了使程序简化，你可在DATA语句中使用字母C，P，或J来表示音乐的类型。你可以这样写：

```
1000 trpe$ =array$(row, 3)
1010 IF type$ = "C" THEN type$ =Classical"
1020 IF type$ = "P" THEN type$ = "Pop"
1030 IF type$ = "J" THEN type$ = "Jazz"
1040 array$(row, 3)=type$
```

用FOR…NEXT循环填两维数组是较容易的。通过上

例，我们看到有四行，每行有 3 项，因此可用下程序装入数组：

```
100 FOR row = 1 TO 4
110 FOR column = 1 TO 3
120 READ array $(row, column)
130 NEXT column
140 NEXT row
```

在这个技术中，依赖于对DATA语句中行数的计数，而不是仅仅设置数字标志的结束。

RESTORE

与READ有关的一个很有用的功能是你可以把数据指针重新设到DATA语句中的第一个数据。‘RESTORE 1000 ’把数据指针移到1000行的DATA语句的第一个数据。如果没有1000行，则产生错误信息‘Line does not exist ’（此行不存在）。如果1000行不是DATA语句，则向后搜寻，指针移到下一个DATA语句。在RESTORE命令中不能使用表达式，只能使用数字参数，这就限制了它和READ及DATA的结合使用，但它可以允许你定义或重新定义所有程序设计的细节。当程序必须重新运行时，你可以用它装入数组，不同的数据可赋给不同的变量等。把数据存贮在DATA语句中并且也存贮在数组中，则是浪费的。如果只需访问数据，把数据存入DATA语句即可。如果还需对数据进行操作，则应把它们填入数组。

TIME

当Amstrad开通时，一个计数器开始从零计数。这个计数器有 4 个字节（32位），可计的最大数约为 4.3E +

09，计到这个数后再从零开始计。计数器每1/300秒计一次数，你可以用它来做一个准确的时钟。为了对一个事件记时，你必须在事件开始前把时间值存入一个实数变量。当事件结束时，你从现行时间减去这个值，然后再除以 300。注意：在磁带操作期间，时钟不计数。

TIME可以作为数字用。在下例中，它用来对反应速度计时。这个程序先清屏，在一个随机的间隔后，则显示"HIT KEy"（按键）。然后计数显示提示和按键之间的时间并显示出结果。大多数人对此的反应速度约为0.25秒。下面是程序：

```
10 ON BREAK GOSUB 510
20 ON ERROR GOTO 510
30 MODE 0:PEN 7:PAPER 6
40 INK 4, 26, 15:SPEED INK 10, 10:CLS
50 LOCATE 4, 1
60 PRINT "Reaction Timer"
70 DEF FN rand(n)= INT(RND(1)*n)+1
80 LOCATE 2, 5:PEN 3
90 PRINT "When the message";
100 LOCATE 2, 7:PEN 1
110 PRINT "HIT KEY 'appears, "
120 PEN 3
130 LOCATE 3, 9::PRINT "Press any key"
140 PEN 4
150 LOCATE 1, 24
160 PRINT "Press space to start"
```

```
170 IF INKEY(47) = - 1 THEN 170
180 CLS
190 SPEED KEY 1, 1
200 FOR Pause = 1 TO FNrand(10000):NEXT
210 CALL &BB03:MODE 0
220 LOCATE 6, 10:PRINT "HIT KEY"
230 const =TIME
240 a $ =INKY $:IF a $ = " " THEN 240
250 reactiontime = TIME - const
260 goes = goes + 1
270 rtime = reactiontime/300
280 totime = totime + rtime
290 avtime = totime/goes
300 MODE 1 ::LOCATE 10, 10
310 PRINT "Your reaction time was"
320 LOCATE 10, 12: PRINT USING "## . #
    ##" ; rtime;
330 PRINT "seconds" ;
340 LOCATE 10, 15
350 PRINT "Average so far: " ;
360 PRINT USING "##. ###" ; avtime;
370 PRINT "seconds" ;
380 LOCATE 13, 25:PRINT "Another? ...y/n"
390 CALL &BB00
400 encore $ =UPPER $( INKEY $ )
410 IF encore $ = "Y" THEN 180
```

```
420 IF encore $ < > "N" THEN 400
430 CLS
440 LOCATE 6 , 10
450 PRINT "Your aver age reaction time"
460 LOCATE 11, 12
470 PRINT "over" ; goes ; "tries was"
480 LOCATE 10, 15
490 PRINT avtime; "seconds"
500 'Restore keyaobard to normal
510 CALL &BB00:CALL &BB03
```

·几何功能

Amstrad有各种几何功能，这些功能在图形程序中有特殊的价值，如画圆、画螺线等。SIN，COS，TAN，ATN都可以计算（见第8章）。大多数家庭计算机Basic要求处理弧度，而不是角度。但Amstrad允许你用DEG和RAD指令在两者间转换。为了把角度转换为弧度，你可以使用关系式："弧度＝（π/180）*角度"。但Amstrad允许你只使用角度或只使用弧度。

存 贮 器

Amstrad 的存贮器为64K。1K为1024 字节。每个字节可存贮一个0～255之间的数字，对于超出这个范围的数字可使用2个字节或其它技巧。关于存贮器的大多数问题在第7章说明。

PEEK和POKE

这两个命令允许你直接访问Amstrad 的RAM，PEEK

允许你检查存贮单元的内容，POKE允许你把一个值放入存贮器。因此，‘POKE 43800，255’把255放入地为址43800的单元；‘PRINT PEEK（43800）’则显示在行号中给出地址的单元内容。

Amstrad 有两种存贮器：ROM和RAM。ROM是只读存贮器，RAM是随机存贮器。当关机时，ROM中的内容不丢失也不改变。而RAM中的内容则在关机时被清除。

可以把RAM想象为一个个模块，有的用于存贮程序，有的用于存贮数据。还有一块是用于屏幕显示，改变这个模块就可以改变屏幕显示。因此，利用 POKE 也可以改变屏幕显示，利用PEEK可得到屏幕的信息。POKE经常用于把机器码的程序装入RAM，这要利用READ指令从DATA语句中读入机器码指令和数据。

每个存贮单元称为一个地址。Basic程序和数据的最大地址为43907。

注意：你不能用POKE命令把一个大于255的数放入一个存贮地址。PEEK也不能产生一个大于255的数。

数 据 库 程 序

最后，我们举出一个非常简单的程序，它包含了本章中说明的所有概念。它是一个3列数据库，可作为书、磁带的索引。它把数据存入磁带，每次运行程序时重新装入，以便它可适于不同的应用，它作为一个说明程序设计技术的工具，而不是一个完整的程序。你可以改进它以适于自己的需要。下面是这个程序：

```
10 ' Simple Database
```

```
20  'with tape facilities
30  '
40  'Initialisation
50 CLS
60 DIM array $ ( 300, 10 )
70 numberofentries = 0
80 numberofcols = 3
90  '
100 field $( 1 ) = "Name"
110 field $( 2 ) = "Address"
120 field $( 3 ) = "Phone"
130  '
140 FOR i = 1 TO numberofcols
150 find $ =find $ +LEFT $ ( field$ (i), 1 )
160 NEXT
170  '
180 option $ = "LSFEQ"
190  'Load, Save, Find, Enter, Quit
200 submenu $ = "NADM"
210  'Next/Alter/Delete/Menu
220  '
230 REM Entry to Main Menu
240 menucol =17:CLS
250 LOCATE 17, 1 :PRINT "Main Menu" ;
260 LOCATE menucol, 5:PRINT "L......Load"
270 LOCATE menucol, 7:PRINT "S......Save"
```

```
280 LOCATE menucol, 9:PRINT "F...... Find "
290 LOCATE menucol, 11:PRINT "E...Enter "
300 LOCATE menucol, 15:PRINT "Q...Quit"
310 LOCATE 19, 24:PRINT "Select"
320  '
330  ' Clear keyboard buffer
340 GOSUB 1450
350 akey $ =UPPER $ ( INKEY $ )
360 IF akey $ = "" THEN 350
370 option =INSTR ( option $, akey $ )
380 IF option = 0 THEN 350
390 IF option = 5 THEN GOTO 1150
400 CLS
410 ON option GOSUB 450, 560, 670, 1030
420 GOTO 240
430  '
440 'Load
450 OPENIN "datafile"
460 INPUT#9, numberofrows
470 numberofentries=numberofrows
480 FOR row = 1 TO numberofrows
490 FOR col = 1  TO numberofcols
500 IF EOF THEN 530
510 LINE INPUT#9, array# ( row, col )
520 NEXT:NEXT
530 CLOSEIN:RETURN
```

```
540 '
550 'Save
560 IF numberofentries< > 0 THER 590
570 LOCATE 13, 10:PRINT "NO data to
    save"
580 GOSUB 1290:RETURN
590 OPENOUT "datafile"
600 PRINT# 9, numberofentries
610 FOR row = 1 TO numberofentries
620 FOR col = 1 TO numberofcols
630 PRINT# 9, array $(row, col)
640 NEXT:NEXT:CLOSEOUT:RETURN
650 '
660 ' Find
670 ' IF numberofenties = 0  THEN RETURN
680 LOCATE 17, 1:PRINT "Find Menu"
690 FOR i =1 TO numberofcols
700 LOCATE menucol, i * 2 + 5
710 PRINT LEFT $( field$(i), 1 );
720 PRINT "......" ; field $(i)
730 NEXT
740 LOCATE 19, 24:PRINT "Select"
750 GOSUB 1450
760 akey $ =UPPER $( INKEY $ )
770 IF akey $ = "" THEN 760
780 field =INSTR( find $, akey $ )
```

```
790 IF field = 0 THEN 760
800 '
810 CLS:LOCATE 1, 10
820 PRINT "Enter" ; field $ ( field ) ;
830 PRINT "to find" ;
840 INPUT pattern $
850 '
860 'Start on first row
870 row =1
880 afind =INSTR ( array $ ( row, field ),
    pattern $ )
890 'NO find - so do next row
900 IF afind = 0 THEN 970
910  '
920 GOSUB 1380: 'Display
930 GOSUB 1490: 'Sub Menu
940  '
950 IF choice $ = "M" THEN RETURN
960  'Next row
970 row =row +1
980 IF row < =numberofentries THEN 880
990 RETURN
1000  '
1010  'Enter
1020 CLS
1030 numberofentries =numberofentries + 1
```

```
1040 FOR col=1 TO numberofcols
1050 LOCATE 1, Col*2 + 5
1060 PRINT field$(col); "..." ;
1070 LINE INPUT array$(numberofentries,
     col)
1080 NEXT
1090 LOCATE 15, 20:PRINT "More...Y/N ";
1100 GOSUB 1230
1110 IF y.n$ = "Y" THEN 1020
1120 RETURN
1130 '
1140 'Quit-Sure? If not, do menu
1150 CLS
1160 LOCATE 10, 10
1170 PRINT "Are you sure...Y/N"
1180 GOSUB 1230
1190 IF y.n$ = "N" THEN 240
1200 CLS:END
1210 '
1220 ' Get yes/no as upper case Y or N
1230 y.n$=UPPER$(INKEY$)
1240 IF y.n$ = " " THEN 1230
1250 IF y.n$ <> "Y" AND y.n$ <> "N"
     THEN 1230
1260 RETURN
1270 '
```

```
1280  'Press Space routine
1290 LOCATE 1, 24:PRINT SPACE $ ( 38 );
1300 LOCATE 7, 24
1310 PRINT "Press Space Bar to continue" ;
1320 GOSUB 1450
1330 a $ =INKEY $:IF a $ = " " THEN 1330
1340 IF a $ =CHR $(32) THEN RFTURN
1350 GOTO 1330
1360  '
1370  'display a record
1380 CLS
1390 FOR col =1 TO numberofcols
1400 PRINT field $ ( col ), array $ (row, col)
1410 NEXT
1420 RETURN
1430  '
1440  'Clear Keyboard Buffer
1450 CALL &BB03:RETURN
1460  '
1470  'Next/Alter/Delete/Menu
1480  'as Find routine sub-menu
1490 LOCATE 1, 24:PRINT SPACE $(38)
1500 LOCATE 9, 24
1510 PRINT "Next/Alter/Delete/Menu"
1520 GOSUB 1450
1530 choice $ =UPPER $ ( INKEY $ )
```

```
1540 IF choice $ = " " THEN 1530
1550 choice=INSTR ( submenu $, choice $ )
1560 IF choice= 0 THEN 1530
1570  'Menu or Next
1580 IF choice $ = "M" OR choice $ = "N"
     THEN RETURN
1590  '
1600 IF choice $ = "D" THEN GOSUB 1710:
     RETURN
1610  '
1620  'Amend Entry
1630 CLS
1640 FOR col =1 TO numberofcols
1650 LOCATE 1, col * 2 + 5
1660 PRINT "Enter new" ; field $ (col) ; "...
     ...;
1670 INPUT array $ ( row, col )
1680 NEXT:RETURN
1690  '
1700  'Delete row
1710 FOR col =1 TO numberofcols
1720 array $(row, col) = ""
1730 NEXT:CLS:RETURN
```

第五章　设计一个游戏

在本章中我们说明如何用文本和文本命令来设计一个简单的游戏。这个游戏使用了许多前几章描述过的技巧，通过它可使你熟悉程序设计。

在这个游戏中，你在屏幕上移动一条蛇，并吃掉圆点。这条蛇不可以跑出屏幕的边框，也不可回身或跑过它的尾巴，并且这条蛇随着所吃的点数而变长。点在屏幕上随机出现，如果你在点后面走得不够快，则记分为0并且点消失。你也可使用TIME来把这个游戏加上时间限制。

编游戏程序

边界

首先，你要围屏幕划一个边界，这可通过箭头字符来做。当屏幕处于MODE 1时，顶边和底边都需要39个字符长，我们可以使用 STRING $命令。顶边和底边的程序为：

```
240 LOCATE 1, 1:PRINT STRING $( 39,
    241 );
260 LOCATE 1, 25:PRINT STRING $( 39,
    240 );
```

上述的STRING $命令也可写为STRING $(39, CHR $(241)) 和STRING $(39, CHR $(240))。

边界的两边可用FOR…NEXT 循环来做，屏幕的左边用右箭头，屏幕的右边用左箭头：

```
280 FOR row = 2 TO 24
300 LOCATE 1, row PRINT CHR $(243);
310 LOCATE 39, row:PRINT CHR $(242);
320 NEXT
```

边界的上面可用于显示记分：

```
330 LOCATE 15, 1:PRINT "Score = 0";
340 score = 0
```

字符

下面我们需要定义字符，我们用小写的x做为蛇头，用一个图案方块作蛇身：

```
420 top=ASC("x"):body =207
430 head $=CHR $ (top) :body $ =CHR $
    (body)
```

这里我们使用变量的原因是：以后我们将检查是否某些字符处于屏幕上特定位置。

观察屏幕

Amstrad 的 Basic 不能直接提供的功能是屏幕 PEEK。有时，能建立任意给定屏幕单元的 ASCII 码内容是很有用的。我们可以写一个小程序来做这一点，并且把它放在程序的开头：

```
40 MEMORY 43798
50 address = 43800
60 DATA 197, 213, 229, 245, 205, 96, 187
70 DATA 50, 23, 171, 241, 225, 209, 193, 201
```

```
80 DATA 0
90 READ value
100 IF value = 0 THEN 160
110 POKE address, value
120 address =address+1
130 GOTO 90
```

现在，当我们想PEER屏幕时，则设置光标，并发出CALL 43800来运行这段程序。然后，PEEK（43799）将得出光标处字符单元的内容。

移动

下面我们将定义蛇的起始位置和移动方向。首先，我们看看如何在屏幕上移动蛇。

当蛇移动时，慢慢地打印蛇头和蛇身字符是不可能的，但我们可以只保持蛇头和蛇尾的轨迹。我们使用一个两维整数组和两个指针，一个指向头的坐标，一个指向尾的坐标。数组元素‘Snake%（n.1）’给出蛇的列，‘Snake%（n.2）’给出蛇的行。

首先，我们设蛇的长度极限：

```
160  maxlen =300
```

然后设数组的大小：

```
190  DIM snake%(maxlen,2)
```

变量‘head’将指向蛇头的坐标，我们让它先指向Snake%的第一个数组元素，变量‘tail’指向snake%的第三个元素：

```
480 head =1:tail = 3
```

当蛇头移动时，我们要计算它的新坐标，并存入新单

元。如果‘head ’的值小于1，再把它设为‘maxlen ’，以便反向使用‘snake% ’中的元素，使其再回到1。对于‘tail ’也是如此。

开始，我们设蛇长为三单位，并把头、中间和尾的坐标放入数组：

```
510 snake%(head, 1)=20:snake%(head, 2)=12
520  'starting location for head
540 snake%(2, 1)=20:snake%(2, 2)=13
550  'middle section
570 snake%(tail, 1)=20:snake%(tail, 2)=14
580  'last section-tail
```

上面程序中所用的数字使蛇大约出现在屏幕的角落，但你也可以改变。

我们每时都要知道蛇头所在的行和列，为此我们使用4个变量。

```
600 oldcol=snake%(head, 1)
610 oldrow=snake%(head, 2)
640 newcol=oldcol
650 newrow=oldrow
```

最后，我们在蛇的初始位置显示蛇：

```
760 LOCATE snake%(head, 1),snake%(head,
    2)
770 PRINT head$;
790 LOCATE snake%(2, 1), snake%(2, 2)
800 PRINT tail$;
820 LOCATE snake%(tail, 1), snake%(tail, 2)
```

```
830 PRINT tail $;
```

为了移动蛇，我们首先空出现有的蛇尾：

```
930 LOCATE snake%(tail, 1), snake%
    (tail, 2)
940 PRINT "";
```

我们减少蛇尾的指针，以使它移向蛇头：

```
970 tail=tail-1
1000 IF tail=0 THEN tail=maxlen
```

我们也必须替换头、尾或身的字符：

```
1040 LOCATE snake%(head, 1),snake%
(head,2)
1050 PRINT tail $;
```

改变方向

我们使用‘z’、‘x’、‘/’和‘;’键来向左、右、上和下移动：

```
460 keyboard $ = "zx/;"
```

读键盘的命令为：

```
870 akey $ =INKEY $
880 IF akey $ = "" THEN akey $ =lastkey $
```

为了把所按的键转变为蛇头的移动，我们移动右或左键时，则对列进行减或加，移动上下键时，对行进行加减（见图5.1）我们使用这样的语句：

```
IF akey $ = "z" THEN newcol =oldcol-1
IF akey $ = "x" THEN newcol =oldcol+1
IF akey $ = "/" THEN newrow =oldrow+1
IF akey $ = ";" THEN newrow =oldrow-1
```

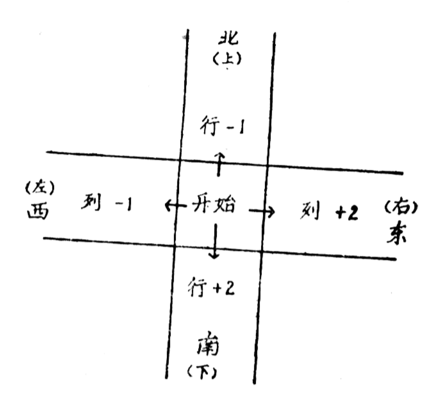


图5.1根据上、下、右、左移动来修改行/列的坐标

但下面的语句可得到更快更好的结果：

1150 newcol＝oldcol＋（akey $＝“z”）－（akey $＝“x”）

1190 newrow＝oldrow－（akey $＝“/”）＋（akey $＝“；”）

读键盘后，我们不马上修改蛇头的坐标，因为我们需要看看新坐标在什么地方。蛇头不能和蛇身重叠，也不能超出边界，所以我们用变量‘newcol’和‘newrow’来暂时保存蛇头的新坐标：

```
1180 IF newcol < 2 OR newcol > 38 THEN 2400
1190 IF newrow < 2 OR newrow > 24 THEN 2400
2400 GOTO 200
```

这里，200行表示进入另一次游戏的起点。

为了测试蛇头是否和蛇身重合，我们用下列语句：

```
1210 LOCATE newcol, newrow
```

```
1220 CALL 43800:check =PEEK ( 43799 )
1230 IF check =top OR check =body THEN
     2400
```

现在我们使蛇头移动到新位置，修改头指针，把新坐标放入数组并修改‘lastkeys ’：

```
1080 head =head - 1
1090 IF head = 0 THEN head =maxlen
1250 PRINT head $;
1280 snake%( head, 1 ) =newcol
1290 snake%( head, 2 ) =newrow
1320 oldcol =newcol
1330 oldrow =newrow
1390 IF INSTR( keyboard $,akey $ ) = 0 THEN
     1450
1420 lastkey $ =akey $
1540 GOTO 870
```

我们必须给出蛇的初始移动：

```
700 lastkey $ = "; "
```

现在你再加入：

```
200 REM
210 MODE 1
```

来完成这个程序并可运行。

产生随机数

此游戏中的一个基本子程序是在屏幕的随机位置产生并显示1—9之间的随机数的程序，为此，我们用DEF FN

定义一个产生随机数的功能：

```
370 DEF FNR(n)=INT(RND(1)*n)+1
```

随机数子程序如下：

```
1580 randnum=FNr( 9 )
1600 randcol =FNr(36)+1
1620 randrow =FNr(22)+1
1660 randnum$=STR$(randnum)
1680 randnum$=MID$(randnum$, 2, 1)
1700 LOCATE randcol, randrow
1720 CALL 43800:check=PEEK(43799)
1730 IF check=body OR check=top THEN
     1600
1740 PRINT randnum$;
1780 RETURN
```

为了跟踪随机教，我们使用一个有0或1的二值'flag'来表示随机数是否在屏幕上。我们设flag的初始值为0：

```
1450 IF randflag=0 THEN GOSUB 1580
```

我们也必须在随机数子程序中加入一行把'randflag'设为1来指出正在显示一个数：

```
1760 randflag=1
```

为了知道蛇是否移到一个随机数处，如果如此，则改变计分，我们在蛇的移动程序中加入：

```
1350 IF newcol=randcol AND newrow=
randrow THEN GOSUB 2000
```

减 少 数 字

游戏的一部分是：随着时间增加出现的教字减少。为此，我们需要一个随时调用的子程序：

```
1500 IF randflag =1 AND Fnr(20) < 3 THEN
GOSUB
1820
1820 randnum = randnum - 1
1840 IF randnum > 0 THEN GOSUB 1660:
RETURN
1880 LOCATE randcol, randrow
1890 PRINT " ";
1910 randflag = 0
1930 randcol = 0 : randrow = 0
1950 RETURN
```

记分

记分子程序本身是十分简单的。我们只需把随机数加到记分上，显示新记分并把随机数标志置 0 以表示屏幕上没有数：

```
2000 score = score + randnum
2020 LOCATE 22, 1:PRINT score;
2040 randflag = 0
2050 randcol = 0 : randrow = 0
2370 RETURN
```

为了使蛇随着吃的点教变长，我们使用蛇移动程序，但去掉“删除尾”部分；

```
2080 FOR inc =1 TO randnum
2150 akey $ =inkey $
2160 IF akey $ = "" THEN akey $ =lastkey $
2170 newcol =oldcol + ( akey $ = "z" ) -(akey $
 = "x" )
2180 newrow =oldrow - ( akey $ = "/" ) +
( akey $ = "; " )
2190 IF newcol < 2 OR newcol > 38 THEN 2400
2200 IF newrow < 2 OR newrow > 24 THEN 2400
2210 LOCATE snake%( head, 1)snake%(head,
 2 )
2220 PRINT tail $;
2230 LOCATE newcol, newrow
2240 CALL 43800:check =PEEK ( 43799 )
2250 IF check =top OR check =body THEN
2400
2260 PRINT head $ ;
2280 head =head -1
2290 IF head = 0 THEN head =maxlen
2300 snake%( head, 1 ) =newcol
2310 snake%( head, 2 ) =newrow
2320 oldcol =newcol
2330 oldrow =newrow
2340 IF INSTR(keyboard $, akey $) = 0 THEN
2360
2350 lastkey $ =akey $
```

```
2360 NEXT
2370 RETURN
```

现在你有了一个完整的程序，你也可很容易地修改它。下面是这个程序的全部：

```
10 REM ++++++++Snake++++++++
20  'Set up machine code routine
30  'to PEEK the text screen
40 MEMORY 43798
50 address = 43800
60 DATA 197, 213, 229, 245, 205, 96, 187
70 DATA 50, 23, 171, 241, 225, 209, 193, 201
80 DATA 0
90 READ value
100 IF value = 0 THEN 160
110 POKE address, value
120 address = address + 1
130 GOTO 90
140  '
150  'Set up snake array
160 maxlen = 300
170  'maxken is longest snake can be
180 DIM snake%(maxlen, 2)
190  'snake array, holds coords of head
200 MODE 1
210  '
220  '    Draw a Border
```

```
230 LOCATE 1, 1:PRINT STRING$(39, 241);
240 '     Top line
250 LOCATE 1, 25:PRINT STRING$ ( 39,
    240 );
260 '     Bottom line
270 FOR row = 2 TO 24
280 '     Sides-Left, Right
290 LOCATE 1, row:PRINT CHR$ ( 243 );
300 LOCATE 39, row:PRINT CHR$(242);
310 NEXT
320 '
330 LOCATE 15, 1:PRINT "Score = 0 " ;
340 score = 0
350 '———Set up variables etc———
360 '
370 DEF FNr(n)=INT( RND(1 ) *n ) +1
380 'user function-generates random
390 'numbers in the range 1 to n
400 'and tail:, 1 is col, 2 is row
410 '
420 top=ASC( "x" ):body =207
430 head$ =CHR$ ( top ):tail$ =CHR$(body)
440 'snake characters
450 '
460 keyboard$ = "zx/; "
470 'key controls-left, right, up, down
```

```
480 head=1:tail=3
490  'head & tail point to array snake%
500  '
510 snake%( ead, 1) =20:snake%( head, 2 )
12
520  'starting location for head
530  '
540 snake%(2, 1) =20:snake%(2, 2) =13
550  'middle section
560  '
570 snake%( tail, 1) =20:snake%( tail, 2) =
14
580  'last section-tail
590  '
600 oldcol=snake%( head, 1)
610 oldrow=snake%( head, 2)
620  'oldcol/row are col/row coords
630  'for snake 's head
640 newcol=oldcol
650 newrow=oldrow
660  'newcol/row are for updating
670  'head coords-see main routine
680  '
690  '
700 lastkey$ = ", "
710  'snake starts heading up the screen
```

```
720 '———End of Definitions———
730 '
740 '    Draw the Snake
750 'Print snake: -head, middle & tail
760 LOCATE snake%( head, 1 ) snake%
( head, 2 )
770 PRINT head$;
780 '
790 LOCATE snake%( 2, 1 ), snake%( 2, 2 )
800 PRINT tail$;
810 '
820 LOCATE snake%( tail, 1 ), snake, 2 )
830 PRINT tail$;
840 '
850 'Now start the game itself
860 '!!!!!!!!!Main Routine!!!!!!!!!
870 '
880 akey$ =INKEY$
890 IF akey$ = "" THEN akey$ =lastkey$
900 ' if no key pressed, carry on in
910 'previous direction
920 '
930 LOCATE snake%( tail, 1 ), snake%
( tail, 2 )
940 PRINT "";
950 ' erase tail
```

```
960 '
970 ' tail=tail-1
980 ' decrement tail pointer
990 '
1000 IF tail=0 THEN tail=maxlen
1010 ' force tail to far end of array
1020 ' if tail points to zero
1030 '
1040 LOCATE snake%(head, 1), snake%
(head, 2)
1050 PRINT tail$;
1060 ,
1070 ' replace head character with ta'
1080 head=head-1
1090 'decrement head pointer
1100 IF head=0 THEN head=maxlen
1110 ' force to far end if points to start
1120 ' Update row and column values
1130 ' is=location of head
1140 ' Using Boolean logic
1150 newcol=oldcol+(akey$ =" z" )-(akey$
$ = "x" )
1160 newrow=oldrow-(akey$ = "/" ) +
(akey$ = "; " )
1170 '
1180 IF newcol < 2 OR newcol > 38 THEN 2400
```

```
1190 IF newrow < 2 OR newrow > 24 THEN 2400
1200  ' if off screen, end of game
1210  LOCATE newcol, newrow
1220 CALL 43800:check =PEEK ( 43799 )
1230 IF check = top OR check = body THEN2400
1240  ' Run into self
1250 PRINT head $;
1260  ' redraw head in new location
1270  '
1280 snake% ( head, 1 ) =newcol
1290 snake% ( head, 2 ) =newrow
1300  'update coords in snake array
1310  '
1320 oldcol = newcol
1330 oldrow = newrow
1340  '
1350 IF newcol = randcol AND newrow = rand-
 row THEN GOSUB 2000
1360  'hit the random number
1370  ' so do score routine
1380  '
1390  ' IF INSTR ( keyboard $, akey $ ) = 0
 THEN 1450
1400  ' if no valid key pressed,
1410  ' leave lastkey$ as is
1420 lastkey $ =akey $
```

```
1430   ' set lastkey pressed to current key
1440   '
1450 IF randflag = 0 THEN  GOSUB 1580
1460   ' randlag is  zero if no random
1470   ' number is on the screen
1480   ' if it's zero - Place a new one
1490   '
1500 IF randflag =1  AND FNr ( 20 ) < 3 THEN
 HOSUB 1820
1510   'decrement any random number
1520   ' at random moments
1530   '
1540 GOTO 880
1550   ' repeat main routine
1560 REM————Random numbers routine
1570   '
1580 rendnum =FNr ( 9 )
1590   ' randnum  is 1  to 9
1600 randcol =FNr ( 36 ) +1
1610   ' randcol is  2  to 37
1620 randrow =FNr(22)+ 1
1830   ' randrow is 2  to 23
1640   '
1650   ' set cursor
1660 randnum$  =STR$ ( randnum )
1670   ' convert randnum to a string
```

```
1680 randnum$ =MID$ ( randnum$, 2, 1 )
1690  ' strip leading and trailing spaces
1700 LOCATE randcol, randrow
1710  ' lodate cursor
1720 CALL 43800:check =PEEK ( 43799 )
1730 IF check =top OR check =body THEN
1600
1740 PRINT randnum$;
1750 ,  display randnum
1760 randflag =1
1770  ' set randflag - now have number on screen
1780        RETURN
1790  '———End of Subroutine———
1800  '
1810  ' Decrement random number routine
1820 randnum =randnum - 1
1830  'take one from randnum
1840 IF randnum > 0 THEN GOSUB 1660:
RETURN
1850  ' if it's not zero, gosub display
1860  ' random number routine & return
1870  '
1880 LOCATE randeol, randrow
1890 PRINT " ";
1900  ' erasse randnum
1910 randflag = 0
```

```
1920  ' set flag to zero( no randnum )
1930 randcol = 0 :randrow = 0
1940  ' null these coords
1950        RETURN
1960  '————End of Subroutine———
1970  '
1980 REN. increase length of snake
1990  '
2000 score =score+randnum
2010  ' uPdate score
2020 LOCATE 22, 1:PRINT score;
2030  ' and Print it
2040 randflag = 0
2050  ' reset randflag( no randnum now )
2060 randcol = 0 : randrow = 0
2070  'reset coords
2080 FOR inc =1 TO randnum
2090  ' for the value of the score
2100  ' do the following
2110  ' most of this a rePeat of
2120  ' the main routine
2130  ' and allows the snake to grow
2140  ' to a maximum of maxlen units
2150 akey$ =INKEY$
2160 IF akey$ = " " THEN akey$ =lastkey$
2170 newcol =oldcol+ ( akey$ = "z" ) -
```

```
(akey$ = “x” )
2180 newrow=oldrow-(akey$ = “/” )+
(akey$ = “; ” )
2190 IF newcol<2 OR newcol>38 THEN 2400
2200 IF newrow<2 OR newrow>24 THEN 2400
2210 LOCATE snake%(head, 1), snake%
(head, 2)
2220 PRINT teil $;
2230 LOCATE newcol, newrow
2240 CALL 43800:check=PEEK(43799)
2250 IF check=top OR check=body THEN
2240
2250 IF check=top OR check=body THEN
2400
2260 ’ Run into self
2270 PRINT head$;
2280 head=head-1
2290 IF head=0 THEN head=maxlen
2300 snake%(head, 1)=newcol
2310 snake%(head, 2)=newrow
2320 oldcol=newcol
2330 oldrow=newrow
2340 IF INSTR(keyboard$, akey$)=0
THEN 2360
2350 lastkey$=akey$
2360 NEXT
```

```
2370         RETURN
2380 ———END of Subroutine———
2390  '
2400 GOTO 200
2410  ' run from start if snake's run
2420  ' off the screen
```

第六章　数字和逻辑

表示法

为了利用程序更好地控制计算机，你应熟习三种记数系统：十进制、二进制和十六进制。它们的操作规则是相同的，所以很容易懂。

这三个系统都是使用教学的指数规定。一般的指数表示是在一个数字的右上角写一个小的数字，如3^2。但Amstrad不这样表示.它使用上箭头符号.例如：4↑2表示4的平方。

十进制

和大多教计算系统一样，十进制依赖于位的概念：3在30和300中有不同的值。数字的位置称为个、十、百、千等。如果我们从右到左表示这些位置，则可得到10↑0，10↑1，10↑2，10↑3等，即各位都可用10的几次幂来表示。这样，计算出各位的值，再把它们加到一起，便得到一个数的值。例如：952可这样表示：

2x(10↑0)+5x(10↑1)+2x(10↑2)=925

我们在这些计算中使用10，是因为10进制系统只有0～9这十个不同的数，所以我们以10为基本计数。

二进制

在二进制系统中只有0和1两个数。使用二进系统是因为微机可以使用2值操作。二进数系统中，每位的值为2的几次幂。因此，二进数10表示：0+1x(2↑1)=2(十进表示)。每个二进教位只有两个值，所以以2为基来计算各位的值。每个存贮单元(1字节)可放8个二进数位，下面是

0 ～7 位的值表：

数位	十进值
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Amstrad有把十进制数转换为二进制教的命令BIN $、'PRINT BIN $（9）'显示出1001。你也可使用两个参数来定义要显示多少位。例如：'PRINT BIN $（9、8），则显示00001001。如果你省略了一个参数，Amstrad则产生必要的位数.你可使用的最大数为65535,最小数为－32785。BIN $最多可产生16 位数：'PRINT BIN $（－1）'产生16 个1。如果你定义的位数大于16，则按应该的位数来显示。注意：所得到的结果是一个字符串，不是一个值，还需用MID $，‹ EFT $ 和RIGHT $ 进行处理。

十六进制

汇编语言多使用16进制表示法，缩写为'hex'。它以16为基，用字母A～F表示十进制的 10～15。每个数位的值通过16的幂来计算。下面是一个16进制记数表：

十进制	十六进制	十进制	十六进制
0	0	17	11
1	1	18	12

2	2	19	13
3	3	20	14
4	4	21	15
5	5	22	16
6	6	23	17
7	7	24	18
8	8	25	19
9	9	26	1A
10	A	27	1B
11	B	28	1C
12	C	29	1D
13	D	30	1E
14	E	31	1F
15	F	32	20
16	10		

Amstrad 用 HEX $ 命令把十进数转换为十六进数。'PRINT HEX $（249）'产生F9。它不能处理大于65535的数。

为了方便，我们用'&'表示 16 进数转换为十进数。'PRINT & FF'将产生255。

HEX $ 是一个字符串功能，因此必须用字符串操作来处理结果。

Amstrad 仅用16位整数来进行16进数的处理，如果你使用一个很大的十进数，将得到一个负数。'PRINT & 7FFF'产生32767，'PRINT & 8000'产生－32768。为了避免这一点，你可以定义一个自己的功能来把16进数转换

成可用的十进数。例如：‘DEF FN hx(n) = -（65536 *（n<0）- n）’，此时‘PRINT FN hx（&FFFF）’将产生65535，而不象‘PRINT & FFFF ’产生-1。

布尔逻辑

布尔逻辑是一种处理数字的方法，它包括AND，OR，NOT以及非常复杂的操作。

AND经常用于测试两个条件：如果两者都满足，则产生某个动作。OR 则是两个条件中只要有一个满足，则产生某个动作。

AND，OR和NOT在Basic程序中，通常和IF……THEN一起使用：

```
1010 IF lives < =2 AND SCORE > 500 THEN
PRINT "You're not doing very well so far"
```

通常用括号把条件括起来，以易读一些：

```
100 IF (lives < =2) AND (SCORE < 500) THEN
PRINT "You're not doing very well so far"
```

你可以画一个‘真值表’来使布尔操作易懂。例如：‘IF（count = 3 AND（value = 10）THEN GOTO 2000’的真值表如下：

count = 3 ?	value = 10?	GOTO 2000?
YES	YES	YES
YES	NO	NO
NO	YES	NO
NO	NO	NO

仅当count = 3 和value = 10两者都成立时，200 行才得执行。

如果把上语句改为：‘IF（count = 3）OR（value = 10）THEN GOTO 2000’，则真值表如下：

count = 3？	ualue = 10？	GOTO 2000？
YES	YES	YES
YES	NO	YES
NO	YES	YES
NO	NO	NO

当有一个条件为真时，则执行2000行。

真和假

在我们使用YES和NO的地方，布尔逻辑使用真和假，而计算机使用－1和0。

使用真假关系可使程序员缩短程序。例如：‘IF varible = 3 THEN flag = －1 ELSE flag = 0’可变为‘flag =（varible = 3）’。

你可使用这个技术替换多个IF…THEN 语句，下面两个程序是相等的：

```
999 REM IF…THEN version
1000 IF akey$ = "z" THEN xcoord-1
1010 IF akey$ = "x" THEN xcoord+1
1020 IF akey$ = "; " THEN ycoord-1
1030 IF akey$ = "-" THEN ycoord+1
999 REM Boolean version
1000 xcoord=xcoord+(akey$ = "z") -
(akey$ = "x")
1010 ycoord=ycoord+(akey$ = "; " = "-")
```

使用这样的布尔逻辑产生相当难读的程序，但执行起来

却比IF…THEN语句快。

位测试

布尔操作可以用于测试字节中数字位的位置。例如：‘PRINT 8 AND 2 ’我们得到的结果为0。因为2的二进制表示法为00001000，2的二进制表示法00000010。当在两个数之间使用AND时，当且仅当两个数的同一位都为1，则则结果中的这位也为1。这个规则表示如下：

AND

第一位	第二位	结果
0	0	0
0	1	0
1	0	0
1	1	1

对于8和2我们使用这个规则，则有：

00001000（8）

00000010（2）

00000000　结果＝0

OR的规则为：只要一个操作数的某位为1，则结果的这位为1：

OR

第一位	第二位	结果
0	0	0
0	1	1
1	0	1
1	1	1

‘8 OR 2 ’的结果为：

00001000（8）

00000010（2）

00001010　结果＝（10）

还有一个操作为XOR，它类似于OR，不同点在于：两个操作的同一位都为1时，结果中的这位却为0。它的规则如下：

XOR

第一位	第二位	结果
0	0	0
0	1	1
1	0	1
1	1	0

位分配

我们可以使用二进制逻辑来利用一个字节存贮大量信息。让我们看一个例子。我们假定一个中学有一个数据库，老师需要知道一个学生正在上哪一门课。每个学生可上一对课中的一门课：法语/德语，物理/化学，地理/历史，木工/金属工，生物/地质。每个学生可以是一个男孩或一个女孩，可以在一个高智力组或一个低智力组，并且可以安排'O'级或CSE级考试。因此，有8个不同的对。我们用一个字节表示一个学生所对应的所有对。例如：女学生的字节在第二位为0，而男学生为1。这个字节的每位都对应一个十进制值。如果这个字节为10100111则有值128＋32＋4＋2＋1＝167。下面我们说明如何用这个方法进行编码、译码。

后面我们给出了使用这样的位分配技术进行编和译码的程序。我们把课名和其它标志放在一个两维串变量'attri-

bute$ （8，2）'中，共有8个标志，每个标志有0、1两个值，如：男为1，女为0。下面为这个程序：

```
10 'Bit Mapped data
20 '
30 DATA 0 level, Upper, Female, Biology,
woodwork, Geog raphy, Physics, French
40 DATA CSE, Lower, Male, Geology, Metal-
work, History, Chemistry, German
50 '
60 ' Load Arrays
70 DIM attribute$ ( 8 , 2 )
80 FOR attribute = 1 TO 8
90 READ attribute$ ( attribute, 1 )
100 NEXT
110 '
120 FOR attribute = 1 TO 8
130 READ attribute$ ( attribute, 2 )
140 NEXT
150 '
160 ZONE 20
170 'MAIN MENU
180 CLS: LOCATE 17, 1:PRINT "SELECT"
190 LOCATE 10, 7
200 PRINT "A···Assign a value"
210 LOCATE 10, 9
220 PRINT "C···Check a value"
```

```
230 LOCATE 10, 11
240 PRINT "E···End"
250  '
260 akey$ =UPPER$ (INKEY$ )
270 choice= - (akey$ = "A" ) - 2 * (akey$
= "C" ) - 3 * (akey$ = "E" )
280  '
290 IF choice= 0 THEN 260
300 ON choice GOSUB 350, 460
310 IF choice= 3 THEN CLS: END
320 GOTO 180
330  '
340  'Data Entry
350 CLS:byte= 0
360 FOR each.bit = 0 TO 7
370 PRINT attribute$ (each.but + 1, 1 );
 "···Y/N" ,
380 GOSUB 570
390 IF akey$ = "Y" THEN byte=byte+2^
each.bit PRINT attribute$ (each.bit+ 1 , 1 )
ELSE PRINT attri bute$ (each.bit+1, 2 )
400 NEXT
410 PRINT:PRINT "Value=" ; byte
420 GOSUB 620
430 RETURN
440  '
```

```
450  ' Check a value
460 CLS:PRINT "Enter value"
480 FOR each.bit = 0 TO 7
490 LOCATE 15, each.bit + 7
500 dec.val = 2 ^each.bit
510 IF(byte AND dec.val) =dec.val THEN
    PRINT
attribute$ ( each.bit+ 1, 1 )ELSE PRINT
attribute$ ( each.bit +1, 2 )
520 NEXT
530 GOSUB 520
540 RETURN
550  '
560  ' Y/N Subroutine
570 akey$ =UPPER$ ( INKEY$ )
580 IF akey$ < > "Y" AND akey$ < > "N"
THEN 570
590 RETURN
600  '
610  'Press space subroutine
620 LOBATE 10, 24
630 FRINT "Press space to continue"
640 IF INKEY(47) = - 1 THEN 640
650 RETURN
```

数字处理函数

Amstrad 有若干特殊函数，可使数字处理较容易。下面介绍一些较常用的。

MOD（模）

MOD 用于建立一个数对于另一个数的模，它有点类似于除法中的余数。MOD 回送一个数除以另一个数后所余的整数值。下表可以使你对MOD有一个较清楚的概念：

3 MOD 1	0
2 MOD99	2
3 MOD 7	3
99MOD 1	0
1 MOD 3	1
5 MOD 2	1
99MOD10	9
83MOD40	3

象你看到的一样，许多数 MOD 1 后，结果为0，因为任何整数除 1 后都没有余数。

MIN和MAX

MIN和MAX 从一列数或表过式中回送最小数或最大数，‘Smallest＝MIN（1、2、3、4）’把1赋给变量Smallest。你也可以使用变量和表达式，如：‘biggest＝MAX（number 1，value 3，count）’。

ABS

ABS除掉数值或表达式的负号。例如：‘PRINT ABS（－20）’产生20，‘PRINT ABS（99—199）’产

生100。

SGN

SGN回送－1，0或1这三个值之一。可用此来测试数字、变量或表达式是正、负还是0。例如：'expr.sign＝SGN（33－39）'把－1赋给变量'expr.sing'，因为33－39产生一个负数。如果结果为0，则回送0，如果结果为正数，则回送1。

PI

PI是一个系统常数，就是我们称为π的圆围率。PI经常在三角学中计数圆的面积等。在一个功能定义中，我们经常这样定义圆的面积：

```
10 DEF FN（area）＝PI＊radius^2
```

LOG，EXP和LOG10

LOG(n)回送数n的自然对数，这个对数是以e为底的。

EXP（n）为反对数，它回送e^n，因此EXP（LOG（n）回送n。

LOG10（n）回送以10为底的数n的对数。对数使大数的处理较简单，例如：两个大数相乘的一个较快的方法是，先查两个数的对数，把它们加在一起，然后由反对数得出结果。

CINT和CREAL

这两个函数把表达式转换为不同的数据类型。CINT把数转变为整数，CREAL把数转变为实数。'CINT（9、999）'产生10。CREAL可这样使用：

```
10 x%＝9
20 x%＝x%＋0.5
30 PRINT CREAL（x%＋0.5）
```

UNT

有两个方法可处理10进数的16位（两字节）表示法。你可使用全部16位来表示十进数，此时的数字定围为0～65535。标准的方法是：任意十进数位的值可由2的幂来表示，这些位是从右到左计数。

但是，在这个系统中你不能表示负数。一个可行的方法是用第15位做标志，来表示负数。这时可表示的范围为－32768～＋32768。负数可用2的补码进行转换。

UNT把一个无符号的十进数转换成有符号的2的补码。这用于汇编语言程序设计的计算需要。

第七章　机　器　码

机器码是计算机内部使用的语言。它很难学，但执行起来很快。机器码没有象LOCATE或PRINT这样的命令，它需要写一个程序来执行这样的指令，然而其执行速度迅快于Basic。

寄　存　器

机器码使用寄存器。寄存器用字母A～H 和L来表示，并且有一些命令令可一次使用 2 个寄存 器，寄 存 器 对 为AF，BC，DE和HL。A寄存器是最常用的，‘A ’为标 准的累加器。

机 器 码 指 令

在Z80 机的指令集合中有这样的指令：把数值装入寄存器、把寄存器的内容存入主存等。

OP—cades（操作码）

机器码程序是一组字节。每条指令或命令有一个对应的数字表示，并把这些数称为操作码。例如：这样两字节的数0000011010000000（十进数 6 和128）表示把128 装入B寄存器。第一个字节是‘装入 B ’指令，第二个字节表示 操 作数。

助记符

机器码和二进数从表面上看来是很难区分的，而 且 也很难记忆。但通过使用汇编语言可解决这个问题。汇编语言

利用缩写符号来表示机器码，前面那个例子可用‘LD B，n ’来表示。这样的缩写符号称为助记符。汇编语言程序由助记符和数据组成，一个叫做汇编的程序可以把它们转变为机器码，并存入RAM。

Basic和机器码混用

你不必用汇编语言写整个程序。Basic有一个非常有用的特性，就是在Basic程序中，你可以调用机器码子程序。你可以调用在驻存在ROM中的程序来完我一些 Basic 不可能做的操作。

机器码程序装入主存后，可以通过Basic命令‘CALL ’再加上一个地址来使机器执行这个程序。此时，机器将暂停Basic程序，跳到给出的RAM地址去执行机器码指令。如果机器码程序的末尾有一个助记符‘RET ’，则返回调用程序。从Basic程序调用机器码程序的过程与 GOSUB……RETURN类似。

机 器 码 程 序

我们将给出一些你可在Basic 程序中调用和使用的机器码程序，研究这些程序，将帮你学习汇编语言程序设计。在这之前，我们先讲几个有关Basic/机器码连接的几个重要问题。

因为Basic程序及其数据要占用RAM内存贮空间，因此我们必须告诉Amstrad保留一些存贮器，以便Basic不能用这些存贮器。如果我们不这样做，数组中的数据存贮可能复盖机器码程序。这个保留操作是由MEMORY命令再加上一

个地址来完成的。这就使程序不能使用高于给定地址的存贮空间，因此可把机器码程序从这个地址开始存放。如果我们希望一个机器码程序从地址43880开始，则可用命令‘MEMORY 43879 ’。当机器码程序装入存贮器后，我们用 CALL 43880来使这个子程序执行。

卷动屏幕

第一个机器码程序是最简单，它允许你把屏幕显示向上或向下移动一行，如果使用FOR…NEXT 循环来调这个程序，则可按你要求卷动屏幕。

Amstrad 的 ROM 中有若干机器码程序，它们不能用Basic命令直接调动。其中一个程序叫做SCR HW ROLL用于移用屏幕的显示行，它的起始地址为&BC4D（十进制为50395）。屏幕卷动的方向依赖于B寄存器的内容。如果 B 寄存器为0，则屏幕下卷，否则，屏幕上卷。新行的颜色取决于寄存器A，A通常为0，即默认的深蓝色。

为了使屏幕卷动，首先要把数值装入B寄存器，这条指令的助记符为‘LD B，#n’。这里n是装入B中的数，“#”表示寄存器是装入给定的值。LDB的操作码为6，这是机器码程序的第一个字节。我们将从地址43880开始汇编，因此我们用Basic命令“POKE 43880，6”把6装入这个地址。下一个字节是指示卷动方向的数值。为了向上卷，我们使用‘POKE 43881，255’。为了向下卷我们使用‘POKE 43881，0 ’。

这下一步是CALL 这个ROM程序。这个程序的起始地址为 &BC4D（十进制为 50395），所以调用指令为‘CALL 50395。但是我们不能在单字节中表示一个大于255的数，因此

要把这个地地分成两个字节表示。用十六进表示法做这一点很容易。50396的16进数为BC4D，所以低字节为 4D（十进数为77），高字节为BC(十进数为188)。当ROM中的程序执行完后，我们需要返回 Basic 程序，这只需使用助记符'RET'即可。现在，我们有了下面的程序及数据：

助记符/数据	十进数值
LD B, #0	(06, 0)
CALL 40395	(205, 77, 188)
RET	(201)

下一步是把操作数装入存贮器。我们采用十进制数，并把它们放入 DATA 语句，然后把它们读出并放入给定的 RAM 地址。Basic 装入程序如下：

```
10  DATA 6, 0, 205, 77, 188, 201
20  MEMORY 43879:address=43879
30  FOR count=1 TO 6
40  READ value
50  POKE address+count, value
60  NEXT count
```

下面的程序是使用这个机器码程序的表演，并假定，机器码程序已装入存贮器：

```
70  CLS: FOR char=65 TO 89
80  PRINT STRING$ (38.char)
90  NEXT char
100 FOR up=1 TO 10
110 POKE 43881, 255
120 CALL 43880
```

```
130 NEXT up
140 FOR down=1 TO 10
150 POKE 43881, 0
160 CALL 43880
170 NEXT down
180 GOTO 100
```

一个较复杂的程序

下个程序较复杂一点，但基本上完成与前面讲过的程序相同的工作，但它可通过把不同的值装入43870单元来使卷出的行有不同的颜色。你也可通过把不同的值装入43871 单元来规定卷动的行数。

首先，我们列出这些机器码的地址、助记符，16进码和十进值：

地址	助记符	操作码/数据	十进值
43870	LD B, #A	06	6
43871		A	10
43872	LD A, #0	3E	62
43873		0	0
43874	PUSH BC	C5	197
43875	PUSH AF	F5	245
43876	LD B, #FF	06	6
43877		FF	255
43878	CALL &BC4D	CD	205
43879		4D	77
43880		BC	188
43881	POP AF	F1	241

43882	POP BC	C1	193
43883	DJNZ	10	16
43884		F5	245
43885	RET	C9	201

在这个程中，B做为一个FOR… NEXT 循环的计数器，使43871的内容指示将要卷动的行号。A寄存器中的值为新行颜色码。地址43874和43875中的指令把BC和AF这两个寄存器对送入堆栈（一个暂存区），因为 SCR HW ROLL程序将占用所有的寄存器。43876中的指令把卷动的方向装入B，我们给出的是上卷。43878的CALL 指令调用 ROM 中的程序，当ROM 程序执行完后，把堆栈中的寄存器对弹出。下一条指令 DJNZ，把B的内容减1，如果不为0，则跳到地址43874。

Basic 装入的程序为：

```
10  MEMORY 43869:address=43869
20  scroll=43870
30  DATA 6, 10,62,0,197,245, 6, 255, 205, 77
40  DATA 188, 241, 193, 16 245, 201
50  FOR count =1 TO 16
60  READ value
70  POKE address+count, value
80  NEXT
90  REM POKE 43871 with colour
100 REM POKE 43873 with number of lines
110 REM POKE 43877 for up/down
120 REM CALL 43870 to scroll
```

观察屏幕上的文本

下一个机器码程序将告诉你在屏幕光标处任意字符的ASCII码。

这个程序在 ROM 中的地址为 47968（十六进为 BB 60）。它把字符的 ASCII 码放在A寄存器中。

为了使用这个程序，用LOCATE命令把光标移到你想测试的字符处，然后 CAII 43800。在光标处的任何字符的 ASSII 码将放入地址 43799，并且可用 PEEK 命令在 Basic中得到。下面是这个程序的 BASIC 装载程序：

```
10  MEMORY 43798
20  '43799 is used to store results
30  aderess = 43800
40  'Routine starts at 43800
50  '
60  'Decimal machine code
70  DATA 205, 96, 187
80  DATA 50, 23, 171, 201
90  DATA 0
100  '
110  'Machine code loader
120  READ value
130  IF value =0 THEN 260
140  POKE address, value
150  address = address + 1
160  GOTO 120
170  '
```

```
180  'all done
190  REM TO use: place cursor with LOCATE
200  REM then CALL 43800
210  REM then PEEK( 43799 )
220  REM This returns ASCII value of
230  REM the character at the cursor position
240  REM or zero if no character
250  '
260  REM EXAMPLE
270  CLS
280  LOCATE i, 1
290  PRINT "ABCDEF"
300  FOR i=1 TO 6
310  LOCATE i, 1
320 CALL 43800
330  value=PEEK( 43799 )
340  LOCATE 1, 10
350  PRINT "Character" ; i; "i =" ;  CHR $
     ( value )
360  PRINT "ASCII code is" ; value
370  LOCATE 10, 20
380  INPUT "press ENTER for next" , a $
390 NEXT
```

下面是这个程序的汇编码：

地址	助记符	16进码	十进码
43800	CALL 0BB60H	CD 60 BB	205 96 107

```
43803    LD (0AB17H)    32 17 AB    50 23 171
48806    RET            C9          201
```

涂色

虽然可以使用一个 WINDOW 命令建立彩色块，但是很麻烦。然而，在从地址 &BC44 开始的ROM中，有一个叫做SCR FILL BOX 的机器码程序，它可根据A寄存器中的颜色码把字符块涂色。这个程序需要4个值来定义涂色块的左，右，上，下字符位置，这些值分别放在H，D，L和E 寄存器中。

下面给出的程序允许你用5个 POKEs 定义来定义块的颜色及块的四个角。使用你自已的机器码程序的优点是：它操作不依赖于任何屏幕窗口。因此，你可定义和使用Basic课文窗口，也可以给矩形涂以素色或具有某种结构的颜色。这些在Basic 中是不易做到的。

```
10 '     Basic Loader
20 '     For box filling
30 MEMORY 43879
40 address=43879
90 DATA 62, 255, 38, 0, 22, 0, 46, 0, 30, 0,
   205
60 DATA 68, 188, 201
70 FOR count=1 TO 14
80 READ value
90 POKE address+count, value
100 NEXT
110 '====== AII Done ======
```

```
120  '
130  'POKE 43881, colour
140  'POKE 43881, left colour
150  'POKE 43885, right column
160  'POKE 43887, top row
170  POKE 43889, bottom row
180  'CALL 43880 to fill box
190  '
200  '=======Demonstration=======
210  MODE 1
220  colour =43881
230  left = 43883:right = 43885
240  top = 43887:bottom = 43889
250 fill = 43880
260  '
270  texture = 255
280  TLHC =0
290  'TLHC i =Top Left-Hand Corner
300  dc =1
310  'dc i=  TLHC increment
320  '
330  'Set up addresses to define box
340  POKE colour, texture
350  POKE left, TLHC
360  POKE top, TLHC
370  POKE bottom, 24-TLHC
```

```
380  POKE right, 39-TLHC
390  'Change texture
400 texture=texture-10
410  IF texture<0 THEN texture=255
420  'Call box fill
430  CALL fill
440  TLHC=TLHC+dc
450  IF TLHC=12 OR TLHC=0 THEN dc=-dc
460  GOTO 340
```

地址	助记符	操作码/数据	十进码
43880	LD A, #n	3E	62
43881		00	0
43882	LD H, #n	26	38
43883		00	0
43884	LD D, #n	16	22
43885		00	0
43886	LD L, #n	2E	46
43887		00	0
43888	LD E, #n	1E	30
43889		00	0
43890	CALL &BC44	CD	205
43891		44	68
43892		BC	188
43893	RET	C9	201

这个程序本身是很简单的。在地址 &BC44 调用 ROM 程序时，它把相应的值装入相应的寄存器。Basic 程序的30

～100行是装载程序，130～180行给出定义颜色的 POKE 地址和块的上、下、左、右单元的 POKE 地址。210～460行为执行。

使字符有不同的颜色

在本章中使用的最后一个 ROM 机器码程序叫做 SCR CHAR INVERT ，此程序把字符的颜色进行异或（ XO-R ）。它假定，B、C寄存器包含两个要用的颜色，而 H、L 寄存器包含字符的屏幕位置，H中为行，L中为列。

Basic 装载程序包括这样一个操作：打印从A 到X的字符串，然后用这个机器码程序对于一行中的字符随机地选择两种颜色之一。在每行左边显示的两个数是颜色码的随机数。当你看到你所需要的组合时，按 ESC暂停程序，记下这两个值，以便以后用在你自己的程序中。

请看下面的程序：

地址	助记符	十六进码	十进码
43890	LD B，#0	06	6
43891		00	0
43892	LD C，#0	0E	14
43893		00	0
43894	LD H，#0	26	38
43895		00	0
43896	LD L，#0	2E	46
43897		00	0
43898	CALL &BC4A	CD	205
43899		4A	74
48900		BC	188

```
48901     RET          C9        201
10  'Character Inverter
20 'Basic Loader
30  DATA 6,0, 14 0, 38, 0, 46, 0 205, 74 188
40  DATA 201
50  MEMORY 43889:address =43889
60  FOR i=1 TO 12
70  READ V
80  POKE address+i, V
90  NEXT
100  '
110  'POKE 43891 with 1st colour
120  'POKE 43893 with 2nd colour
130  'POKE 43895 with column
140  'POKE 43897 with row
150  'CALL 43890 to invert character
160  '
170  '=====DEMONSTRATION=====
180  DEF FNr(n) =INT(RND(1)*255)+1
190  MODE 0
200  char=65
210  aline=43897:position=43895
220  colour1=43891:colour2=43893
230  value1=1:value2=128
240  FOR row=1 TO 24
250  LOCATE 1, row
```

```
260  PRINT STRING $( 19, char );
270  char =char + 1;
280  NEXT
290  FOR row =0 TO 23
300  LOCATE 1, row + 1
310  PRINT USING "###" ; value1; :PR-
     INT "  ";
320  PRINT USING "###" ; value2;
     :PRINT " " ;
330  POKE colour1, value1
340  FOR column = 9 TO 18
350  POKE aline, row
360  POKE position, column
370  POKE colour2, value2
380  CALL 43890
390  NEXT
400  value1 = FNr( 255 ):value 2 = FNr( 255 )
410  NEXT
420  GOTO 290
```

ROM 调用

下面的表给出了一些有用的 ROM 程序。这些程序没有入口条件，因此可不用设参数来从 Basic 中调用。这里列出主要的系统程序，描述其操作和入口，出口条件。

Keyboard 键盘

&BB80 Initialise Key Management System

&BB03 Reset Key Management System

&BB13 Wait for key-press

&BB06 Get next key press in A register

Display 显示

&BB4E Initialise text VDU system

&BB51 Reset text VDU system

&BBBA Initialise graphics VDU system

&BBBD Reset graphics VDU system

&BB6C Clear current window

&BD19 wait for TV frame flyback

Cassette 磁带

&BC65 Initialise cassette system

&BC6E Start cassette motor

&BC71 Stop cassette motor

Sound 声音

&BCA7 Reset Sound Manager System

&BCB6 Stop all sounds

&BCB9 Restart sounds

第八章　介绍图形

颜　色

Amstrad 有三种屏幕方式可决定屏幕的分辨率和颜色。MODE 0 也称为多色方式，每次可在屏幕上显示27种颜色中的16种。MODE 1是默认方式，它最多可显示4种颜色。MODE 2 有最高的分辨率，但每次只能显示2种颜色。下表给出颜色码：

码	颜色	码	颜色
0	Black 黑	14	Pastel Blue 淡蓝
1	Blue 蓝	15	Orange 橙
2	Bright Blue 浅蓝	16	Pink 粉
3	Red 红	17	Pastel Magenta 淡洋红
4	Magenta 洋红	18	Bright Green 浅绿
5	Mauve 浅绿	19	Sea Green 海绿
6	Bright Red 浅红	20	Bright Cyan 浅绿
7	Purple 紫	21	Lime Green 树绿
8	Bright Magenta 浅洋红	22	Pastel Green 淡绿
9	Green 绿	23	Pastel Cyan 淡青
10	Cyan 青	24	Bright Yellow 浅黄
11	Sky Blue 天蓝	25	Pastel Yellow 淡黄
12	Yellow 黄	26	Bright White 浅白
13	White 白		

分辨率

在 MODE 0，每行有20列； MODE 1每行有40 列，MODE 2每行有80列。每个字符由8×8的点阵组成，但点的大小根据屏幕方式而变。任何屏幕上的最小点称为象素。你可以使用 MODE 2来画细致的图形，这些图形不需要多种图形而需要高分辨率。也可用 MODE 2进行字处理，因为80列的格式对字处理是最方便的。MODE 0 有多种颜色，但作图是很粗的，这个方式最好为年轻的用户显示教学程序等。最灵活的是 MODE 1，它有适中的字符大小和四种颜色。

虽然，屏幕本身的分辨率是640×400，但象素的数量是随屏幕方式而变的。在 MODE 2时，分辨率为640×200，MODE 1 为320×200，MODE 0 为160×200。在这三种方式下，字符的高度是相同的，但宽度不同。

图形程序只要做一些很小的修改，就可在任意屏幕方式下使用，但显示的范围和可用颜色有所不同。

边框颜色

你可以改变屏幕的框颜色和背景颜色或前景颜色。你甚至可以定义两种边框颜色，并使它以一定速率变换。边框颜色由Basic 命令BORDER n 来设置，这里n是颜色码。如果给BORDER 两个参数，则设置了两种边框颜色，并且两种颜色不断变换。为了定义颜色变换的速率，我们使用命令SPEED INK 。它需要2个参数来规定每种颜色（也称墨水）出现的时间。时间单位为 1/50秒。SPEED INK

1.1′表示两种颜色以1/50秒的速率交替出现。时间的最大值为 255（大约5秒），而且两个时间数不必相同。例如：'SP-EED INK 25，200′使第一种颜色出现半秒，第二种颜色出现4秒。

背景颜色和前景颜色

置背景颜色和前景颜色不象置边框颜色那么简单。

INK

为了在纸上写字，你必须选择笔灌入墨水，这就是Amstrad 颜色系统的工作方法。命令INK 允许你把颜色值赋给你想用的墨水。例如，你想把0号墨水设为浅兰，应使用NK 0，14。你要把2号墨水设为绿色，则应使用INK 2，9。

当 Amstrad 开通时，每个墨水（0号～15号）都被赋与一种颜色，但是不同的屏幕方式这些颜色是不同的。每个笔被灌入相同号的墨水，这些颜色是机器的默认值，是可以改变的。在上面的例子中，我们把绿色码赋给2号墨水就是在2号笔中灌入了绿墨水。

MODEO的 PEN/INK 默认颜色赋值

PEN（笔）	默认的颜色码	颜色
0	1	蓝
1	24	浅黄
2	20	浅青
3	6	浅红
4	26	浅白

5	0	黑
6	2	浅蓝
7	8	浅洋红
8	10	青
9	12	黄
10	14	淡蓝
11	16	粉
12	18	浅绿
13	22	淡绿
14	1，24	闪烁蓝，浅黄
15	16，11	闪烁粉，天蓝

MODE 1 的 PEN/INK 默认颜色赋值

PEN（笔）	默认的颜色码	颜色
0	1	蓝
1	24	浅黄
2	20	浅青
3	6	浅红
4	1	蓝
5	24	浅黄
6	20	浅青
7	6	浅红
8	1	蓝
9	24	浅黄
10	20	浅青
11	6	浅红

12	1	蓝
13	24	浅黄
14	20	浅青
15	6	浅红

MODE 2 的 PEN/INK 默认颜色赋值

PEN（笔）	默认的颜色码	颜色
0	1	蓝
1	24	浅黄
2	1	蓝
3	24	浅黄
4	1	蓝
5	24	浅黄
6	1	蓝
7	24	浅黄
8	1	蓝
9	24	浅黄
10	1	蓝
11	24	浅黄
12	1	蓝
13	24	浅黄
14	1	蓝
15	24	浅黄

PAPER

为了设置屏幕的背景颜色，你可以使用命令‘PAPER

n，这里n是墨水（INK）号，不是颜色码。因此，PAPE 3表示把背景颜色设为3号墨水所赋的颜色。当 Amstrad 开通时，背景颜色设为0号笔或墨水的颜色，前景颜色即所用的笔号为1。

PEN

命令 PEN 决定用哪种颜色写字符。'PEN 0' 表示屏幕上的字符将为蓝色。如果你用的笔颜色和纸颜色相同，则在屏幕上看不到任何字符。

你也可以使前景和背景各在两种颜色间变换。这需要给 INK 3个参数，如 INK 1，0，26'，第一个数是定义的墨水号，第二、三个数是使用的颜色码。使用 SPEED INK 可以改变变换速率。

图　　形

Amstrad 可在屏幕上产生图形：即彩色线和点组成的象。你也可对图形设置前景颜色和背景颜色。有若干可用的图形命令，下面将分别介绍。

座标

图形屏幕非常象座标纸，其左下角的座标为（0，0），即0列0行，这点称为原点。这和文本屏幕是非常不同的，文本屏幕的左上形坐标为（1，1）。在任何方式中，所有的坐标都是先给出列，后给出行。

POS 和VPOS

POS用来给出跨越“流”的当前的光标的位置。因此，可用它来得知打印头在打印机的滚筒上走了多远。PRINT “A”；POS（#O)产生'A 2'，因为打印了A以后，光标

是处于这行的第二个位置。POS（#8）告诉你打印头走了多远。POS（#9）告诉你在上一个回车后有多少个字符被送入了磁带。

VPOS 回送光标的垂直位置，因为它对屏幕外的任何流都没有意义，所以其参数必须在0～7之间，如：'ycoord=VPOS（#1）'。

MOVE

使用命令 MOVE和 MOVER 可使图形光标在屏幕上移动。MOVE 需要两个参数：要移到的列和行。MOVE 100，100' 把图形光标移到象素坐标为（100，100）的地方。这个坐标是从原点算起的。屏幕右上角的坐标为(639，399）。

MOVER

MOVER 表示相对移，此命令把所提供的参数加到当前光标的列行值上，然后把光标移到这个位置。如果当前的光标位置是（100，100）则命令 'MOVER 10，-20'把光标移到（110，80）。

你可以把光标移出屏幕而不产生出错信息，例如：你可用'MOVE 1000，1000'。但是如果参数的范围超出-32768～+32768，则产生出错信息'Over flow'。

PLOT

为了在给定的屏幕位置设一个象素，你可使用PLOT命令，此命令至少需要2个参数，以表示象素的坐标。象素的颜色可在第三个参数中给出，这里用的是墨水号。'PLOT 50，100，3'把象素设在（50，100）处，并使用3号墨水的颜色。PLOT 不仅可设指定的点。也把光标移到那点。

PLOTR

PLOTR 在屏幕的相对位置设象素。它把参数加到图形光标的当前坐标上，并在这点设象素。它也可有第三个参数来表示颜色。

DRAW

使用命令 DRAW和DRAWR，可通过移动光标来产生一条线。DRAW 按其及的前 2 个参数把光标移到一新点，并在新点与旧点之间产生一条线，也可用第三个参数定义线的颜色。'DRAW 20，20，1'，将在光标所在处和(20，20)点之间画一条线，并且线为 1 号墨水的颜色。

DRAWR

DRAWR 在当前光标和其相对位置间画线。DRAWR 20，20 表示在光标所在点到光标坐标分别加20 的点之间画线。

TEST和TESTR

TEST 和TESTR 命令告诉你特定象素的颜色。'TEST m，n'产生坐标为(m，n)的象素的颜色码。TESTR 则产生相对于光标的象素颜色码。如果 TETST 或TESTR 产生 0，则给定点的象素为背景颜色。这两个命令可这样使用：

check = TEST 100，100　或　check = TESTR 10，-10。

注意：这两个命令也把光标移到所测试的点。删除象素和线

由 PLOT 和DRAW 在背景颜色上画的点和线可以被删除。例如：

```
10   MODE 0
20   MOVE 0, 0
30   DRAW 639, 399, 4
40   GOSOB 1000:'Pause
50   DRAW 0, 0, 0
60   GOSUB 1000:'Pause
70   MOVE 0, 399:DRAWR 639, -399, 7
80   GOSUB 1000
90   DRAW 0, 399, 0
100  GOTO 20
1000 FOR Pause=1 TO 100:NEXT:RETURN
```

这个程序从原点到右上角画一条蓝色的线，然后，又用背景颜色画回到原点，这样就把这条件删除了 。注意：当 MODE 命令发出时，光标被放在原点。任何没有第三个参数的DRAW 和 DRAWR 命令将使用最后一次定义墨水颜色。

XPOS和YPOS

XPOS和 YPOS 回送图形光标的列行坐标。这对于把光标重设到原来画的图形上是很有用的。它们不做为命令使用，因为它们不移动光标，而是把光标的坐标值赋给变量，例如：'xscoord=xPos：ycoord=yPos'。这个技术的使用在下面的例子中说明。

例　子

下面，我们用本章中介绍的命令做一下练习。我们将做一些象方块和圆这样的简单图形。

方块

为了围着屏幕边缘画一条线，我们可使用两种方法。最慢的一种方法是从屏幕左下角的原点开始，一点一点的画，一直画回到原点。每条线可用一个 FOR…NEXT 循环来做，例如：为了从原点向上画，我们从原点（0，0）开始，画所有y值为0～399之间的点，而x值保持为0。下面是这段程序。

```
10  MODE 2:MOVE 0, 0
20  x=0
30  FOR y=0 TO 399
40  PLOT x, y
50  NEXT
```

为了在屏幕上边画线，我们把y值设为399，使x值从0变到639，然后画每一点：

```
60  y=399
70  FOR x=0 TO 639
80  PLOT x y
90  NEXT
```

为了画右边的线，我们把x值设为639，而使y值从399变到0：

```
100  x=639
110  FOR y=399 TO 0 STEP-1
120  PLOT x, y
130  NEXT
```

最后，我们把y设为0，使x值从639变为0：

```
140  y=0
150  FOR x=639 TO 0 STEP -1
```

```
160  PLOT x, y
170  NEXT
```

另一个方法：定义想连线的点，然后在点间画线。屏幕4个角的坐标分别为（0，0），（0，399），(639，399)，（639，0），只要我们把光标依次移到每点，则可画好线：

```
10  MODE 1
20  PLOT 0, 0
30  DRAW 0, 399
40  DRAW 639, 399
50  DRAW 639, 0
60  DRAW 0, 0
```

这种方法比前一种快得多。

图形子程序

子程序可使程序设计变得容易。如果我们有一个需要画许多方块的程序，每次都把画方块的部分写出来是很麻烦的。但是我们可用一个通用的画方块子程序，每当我们需要在屏幕的任意点画方块时，则可调用这个子程序。然后光标回到调用前的位置。我们在这个子程序的入口用 xPos 和 yPos 登记光标的位置，在退出这个子程序前，使光标复位。因为方块的边长相同，我们只需告诉子程序方块右上角的光标和边长。这个子程序如下：

```
1000  xcoord=xPos:ycoord=yPos
1010  PLOT leftcol, toProw
1020  DRAWR  sidelength, 0:' toP line
1430  DRAWR  0, -sidelength: 'righ line
```

```
1040  DRAWR  -sidelength, 0:' bottom line
1050  DRAWR  0, sidelength:' lete line
1060  MOVE x Coord, ycoord
1070  RETURN
```

Amstrad 没有画图的内部命令。但是，可以很容易地写一个子程序，来增加这个功能。画圆的过程是相当容易的，它可在一个 FOR…NEXT 循环中从0到360度一步步的执行。在调用这个子程序前，我们必须定义圆心和半径。在圆表面上的每一点的x坐标可通过把角度的Sin值乘以半径，再加到圆心的x坐标上而得出。每点的y坐标由通过把角度的COS值乘以半径，再加到圆心的y坐标而得出。

唯一注意的一点是，你可以选择是以角度还是以弧度为单位来计算。下面是这个程序：

```
1999  REM CIRCLE subroutine
2000  DEG
2010  xcoord=XPOS: ycoord=Y POS
2020  PLOT centrex, centrey
2030  FOR degrees=1 TO 360
2040  xPoint=centrex+radius*SIN( degrees )
2050  yPoint=centrey+radius*COS( degrees )
2060  PLOT xPoint, yPoint
2070  NEXT
2080  MOVE xcoord, ycoord
2090  RETURN
```

填充图形

我们可以修改方块和圆程序，使之可填充所画的图形。

要做的第一件事是在主程序中设两个变量：yes和No，它们对应于－1和０。我们也可使用另一个变量'fillit'，在我们调用子程序以前，对这个变量也置'yes'或'No'。在子程序本身中，我们有一些对图形涂色的命令，如果'fillit＝0'，我们将用一个 GOTO 跳到这些命令的入口。为了达到这个效果，我们把下面几行加到方块程序中：

```
1061  IF fillit=no THEN 1070
1062  FOR row=toprow TO toprow-sidelength
       STEP -1
1063  DRAWR sidelength, 0
1064  NEXT
```

这段程序在方块中从上到下画线。要记住：在调用这个子程序前，必须定义'yes'和'No'：

```
15  yes=-1:No=0
```

在调用前，你也必须对'fillit'置'yes'或'No'：

```
85  fillit=yes
```

下面是对圆程序增加的命令：

```
1061  IF fillit=no THEN 1070
1062  DRAW centrex, centrey
```

这两条命令从圆周上的各点到圆心画线。和方块程序一样，你也必须把 yes 置为－1，把No置为０并对"fillit"置'yes'。

快速作图

有一些快速画圆的方法。首先，我们可对角度、半径等使用整数变量。

ORIGIN

我们可以把原点设到任意列行处。通常，图形操作的原点设为（0，0）。但是命令 ORIGIN 可以把原点设在屏幕或离开屏幕的任意处。然后，我们使用 DRAWR 或MOVER 命令可以对原点做相对移动。

如果我们把原点设为圆的圆心，则画圆周的计算和操作变得很简单也很快。下面的画圆子程序用'r%'做半径，用'xc%'和'yc%'做原点及圆心：

```
1000  DEG
1010  FOR d%=1 TO 360
1020  ORIGIN xc%, yc%
1030  PLOTR r%*SIN(d%), r%*COS(d%)
1040  NEXT
1050  RETURN
```

这个程序不把光标复位到调用前的位置，但你可很容易的把以前讲的复位方法加进去。

为了把这个程序修改成填充圆的程序，把1030 行的 PLOTR改为DRAWR。如果你想改变圆的颜色，只需把 INK 或 PEN 码加入 PLOTR或DRAWR 命令中。

下面的程序在 MODE 0 下画填入所有默认颜色的圆：

```
10  MODE 0
20  i%=1
30  r%=100
40  xc%=320:yc%=2000
50  GOSUB 1000
60  i%=i%+1
```

```
70  IF i%>15 THEN i%=0
80  GOTO 50
999  REM Circle routine starts here
1000  DEG
1010  FOR d%=1 TO 360
1020  ORIGIN xc%, yc%
1030  DRAWR r%*SIN(d%), r%*COS(d%),
      i%
1040  NEXT
1050  RETURN
```

还有更快的画圆方法。下面的画圆程序看起来很长，但却很快。遗憾的是，这个程序所用的算术原理超出了本书的范围。当速度很重要而不在乎占用多少存贮器时，它是一个很有用的子程序：

```
10  REM fast circle
20  DEG:radius%=190:DIM Point(90, 1)
30  ORIGIN 0, 0
40  centrex=320:centrey=200
50  'Fast circle
60  GOSUB 5000:PRINT delayf; "seconds"
70  '
80  PRINT "Press space to continue"
90  IF INKEY(47)=-1 THEN 90
1000 '
110 'Normal circle
120  GOSUB 6000:PRINT "Fast was"; dela-
```

```
    yf; "seconds" :PRINT "This took" ;
    delayn; "seconds"
130 PRINT "Fast to slow ratio =" ; delayf/
    delayn
140 '
150 END
160 '
4999 'Fast circle
5000 CLS:const=TIME:PRINT "Calculat-
     ing"
5010 'Calculate quadrant points
5020 FOR degree%=0 TO 90
5030 PRINT "." ;
5040 Point ( degree%, 0 ) =radius%
     ( degree% )
5050 Point ( degree%, 1 ) =radius% * COS
     ( degree% )
5060 NEXT
5070 '
5080 'Plot all points
5090 CLS
5100 PLOT centrex+point( degree%, 0 ),
5110 FOR degree%=0 TO 90
     centrey+point= degree%, 1 )
5120 PLOT centrex+point( degree%, 0 ),
     centrey-point( degree%, 1 )
```

```
5130 PLOT centreo-Point(degree%, 0),
     centrey-Point(degree%, 1)
5140 PLOT centrex-Point(degree%, 0),
     centrey+Point(degree%, 1)
5150 NEXT
5160 delayf=(TIME-const)/300
5170 RETURN
5180 '
5999 'Normal circle
6000 CL8:const=TIME
6010 FOR i%=0 TO 360
6020 ORIGIN centrex, centrey
6030 PLOTR radius%*SIN(i%), radius%
     *COS(i%)
6040 NEXT
6050 delayn=(TIME-const)/300
6060 RETURN
```

基于同样的原理，我们还有一个更快的方法：

```
10 'Really fast circles
20 'Using eight segments
30 DEG:CLS
40 radius%=150
50 xorigin%=320:yorigin%=200
60 no.steps%=16
70 step.angle=90/no.steps%
80 chord=(radius%*2*PI)
```

```
90  chord=chord/( 360/steP.angle )
100  int.angle=( 180-steP.angle )/2
110  no.units=no.stePs%/2
120  DIM dx( no.units ), dy( no4units )
130  FOR count%=0 TO no units
140  angle=90-count%*steP.angle
150  angle=int.angle-angle
160  dx( count% ) =chord*COS( angle )
170  dy( count% ) = -( chord*SIN( angle) )
180  NEXT
190  xPoint=xorigin%
200  yPPint=yorigin%+radius%
210  PLOT xPoint, yPoint
220 ' 0 to 45 degrees
230  FOR count%=0 TO no.units
240  DRAWR dx( count% ), dy( count% )
250  NEXT
260 '45 to 90 degrees
270  FOR count%=uo.units TO 0 STEP -1
280  DRAWR -dy( count% ), -dx( count% )
290  NEXT
300 '90 to 135 degrees
310  FOR count%=0 TO no.units
320  DRAWR dy( count% ), -dx( count% )
330  NEXT
340 '135 to 180 degrees
```

```
350 FOR count% =no.units TO 0 STEP -1
360 DRAWR -dx(count%), dy(couot%)
370 NEXT
380 '180 to 255 degrees
390 FOR count% =0 TO no.units
400 DRAWR -dx(count%), -dy(count%)
410 NEXT
420 '225 to 270 degrees
430 FOR count% =no.units TO 0 STEP -1
440 DRAWR dy(count%), dx(count%)
450 NEXT
460 '270 to 315 degrees
470 FOR count% =0 TO no.units
480 DRAWR -dy(count%), dx(count%)
490 NEXT
500 '315 to 360 degrees
510 FOR count% =no.units TO 0 STEP -1
520 DRAWR dx(count%), -dy(count%)
530 NEXT
540 END
```

椭　　圆

画椭圆的方法几乎和画圆是一样的。唯一不同的是：你必须使用两个半径，一个用于y坐标，一个用于x 坐标。椭圆是根据高度和宽度之比来描述的。这个比等于1时，就是圆。下面为把画圆程序修改成的画椭圆程序：

```
1000  DEG
1010  xcoord% =XPOS:ycoord% =YPOS
1020  FOR degrees% =1 TO 360
1030  ORIGIN xcentre%, ycentre%
1040  PLOTR xradius * SIN(degrees%,
      yradius)*
COS(degrees%)
1050  NEXT
1060  MOVE xcoord%, ycoord%
1070  RETURN
```

螺　　线

螺线是较难画的。首先，螺线要转几个弯，第二，圆心随转动的角度而增加。然而，这些问题是容易解决的。

下面是一个画螺线的子程序。为了调用这个程序你必须先定义几个变量，'Centrex' 和 'Centrey' 为圆心，'radius' 为开始半径，'no.turns'为转动的圈数：

```
10  centrex =320:centrey =200
20  radius =1
30  no.turns =4
40  GOSUB 1000:REM Draw Spiral
50  END
999  REM Spiral-drawing subroutine
1000  CLS:DEG
1010  FOR degrees% =0 TO 360 * no.turns
1020  ORIGIN centrex, centrey
```

```
1030  xPoint% =radius * SIN ( degrees% )
1040  yPoint% =radius * COS ( degrees% )
1050  PLOTR xPoint%, yPoint%
1060  radius =radius + degrees% '1000
1070  NEXT
1080  RETURN
```

做为最后一个例子，我们给出一个类似画螺的程序，并说明，如何用PLOTP的第三个参数定义屏幕上点的颜色：

```
10   CLS:DEG
20   centrex =320:centrey =200
30   for deginc =10 TO 1 STEP -1
40   no.turns =3
50   for radalt =1000 TO 300 STEP -50
60   MOVE centrex, centrey
70   DEF FN altrad(n) =radius + degrees%radalt
80   radius =1
90   GOSUB 1000
100  NEXT:NEXT
110  END
999  'SPiral Routine
1000  FOR degrees% =1 TO 390 STEP deginc
1010  ORIGIN centrex, centrey
1020  PLOTR radius * SIN ( degree% ), radius
        * COS ( degrees% ), RND ( 1 ) * 14 + 1
1030  radius = radius + radius + FN altrad ( n )
1040  NEXT:RETURN
```

第九章　高级文本和图形

连接文本和图形

在图形的特殊地方打印文本是很有用的。图形的分辨率比文本分辨率细，因此这种方法适于标注图解这样的任务。图形坐标系统也可用于平滑地移动字符。

TAG和TAGOFF

这个命令不立即产生影响，但它连接文本和图形光标，以便光标点阵左上角的象素和图形光标重合。

下面的子程序画一个钟表的盘面，它很象画圆程序，唯一增加的就是 TAG 命令。TAG 把文本光标放到图形光标的位置，TAGOFF 使文本光标回到原处。我们必须以30度为步长进行循环，因为0，30，60，90，…度的位置对应12，1，2，3，等数字位置，下面是这个程序：

```
10  MODE 1
20  xcentre% =320:ycentre% =200
30  radius% =100
30  radius% =100
40  GOSLB 100
50  END
1000  xcord = XPOS:ycoord = YPOS
1010  DEG
1020  FOR degrees% =0 TO 360 STEP 30
```

```
1030  ORIGIN xcentre%, ycentre%
1040  PLOTR radius% * SIN( degrees% ),
      radius% * COS( degrees% )
1050  TAG
1060  PRINT mid$ ( str$ ( hours ), 2, 2, );
1070  TAGOFF
1080  hours = hours+1
1090  NEXT
1100  MOVE xcoord, ycoord
1110  RETURN
```

TAG 指令表示：如果图形光标移动，文本光标也移动。因此，用 TAG和 TAGOFF把 PRINF语句夹在中间的做法是很有用的（见1050行～1070行）。

为了进一步说明 TAG 的使用和这个命令所允许的文本/图形混合方式，我们来看另一程序。在这个程序中，你可用'z'，'x'，'/'和'；'键按不同方问移动十字定位数。按空格键则从屏幕的右下方和左下方向光标的中心画会聚线。下面是这个程序：

```
10  ON BREAK GOSUB 340:ON ERROR GOTO
    340
20  SPEED KEY 1, 1
30  SYMBOL AFTER 249
40  SYMBOL 250, 24, 24, 24, 231, 24, 24, 24
50  ax =7:ay =6
60  INK  1, 3:INK 3, 26
70  MODE 1:curcol =320:currow =175
```

```
80  cursor$ =CHR$(250)
90   'Move cursor
100 PLOT 0, 400, 3
110 MOVE curcol, currow:TAG
120 CALL &BD19: PRINT cursor$; : TAGO
    FF
130 a$ =LOWER$(INKEY$)
140 IF a$ =CHR$(32) THEN GOSUB 250
150 MOVE curcol, currow:TAG
160 CALL &BD19:PRINT CHR$(32); :TAG
    FF
170 curcol=curcol-8*(a$ = "x" ) +8*
    (a$ = "z" )
180 currow=currow-8*(a$ = "; " ) + 8 *
    (a$ = "/;  )
190 if curcol <1 THEN curco!=639
200 if curcol >639 THEN curcol=1
220 GOTO100
230 'Fire
240 x=curcol 1 ax:y=currow -ay
250 MDVE 0, 0
260 DRAW x, y, 1
270 MVOE 39, 0
280 DRAW x, y, 1
290 MOVE 0, 0
300 DRAW x, y, 1
```

```
310  MOVE 539, 0
320  DRAW x, y, 1
330  RETURN
340  CALL  &BB00
```

屏幕上的写操作

通常，当写一个字符去显示时，则这个字符的点阵擦除其位置处的象素。但是，我们可以把屏幕处理转换为透明方式，使屏幕上出现的字符不擦除象素。透明方式允许你更精确的标注图形，因为标注用的字符不能擦掉所在处的图形。

为了进入透明方式，使用下面的指令：

```
PRINT CHR$(22); CHR$(1);
```

返回一般方式则需要：

```
PRINT CHR$(22); CHR$(0);
```

下面是一个在'HELLO'上面显示'GOODBYE'的程序例子：

```
10  MODE 1
20  PRINT CHR$(22); CHR$(1);
30  LOCATE 10, 10: PRINT "HELLO";
40  LOCATE 10, 10: PRINT "GOODBYE";
```

打印 ASCII 码值在32以下的 CHR$，对屏幕处理产生特殊的影响，这些码称为'非打印控制码'。

象素和字符可以使用3个布尔操作 AND、OR和 XOR来产生。这些操作作用于所写的象素和定义的色彩，其结果放在相关的屏幕存贮器中。

为了设置写屏幕的不同方法，你必须先送入一个非打印

制控码。其方法是打印 CHR $（23），后面再跟一个 CHR $（n），这里n是0～3之间的值。0是默认值，表示一般方式。要对象素 XOR，使用'PRINT CHR $（23）；CHR $1；'。AND 使用'PRINT CHR $（23）；CHR $（2）'。OR 使用' PRINT CHR $（23）；CHR $（3）'。

曲线程序

这是一个产生曲线的计算机程序，这个程序利用写屏幕的几种方式来表示它们的区别。因为画任意一边的过程都是相同的，所以仅用一个子程序处理画线：

```
10  REM Curve Stitch-colour version
20  inkcode=2
30  screencode=0
40  inkmask=0
50  xinc=10:yinc=-7
60  CLS
70  '
80  INK 0, inkcode
90  sceencode=screencode+1
100  screencode=screencode MOD 4
110  PRINT CHR$(23); CHR$(screencode);
120  '
130  REM bottom left
140  xEND=1:yEND=1
150  xstart=1:ystart=400
160  GOSUB 380
```

```
170 '
180  REM  toP right
190  xstart =640:ystart =1
200  yend =400:xend =640
210  xinc = - xinc:yinc = - xinc
220  GOSUB 380
230 '
240  REM bottom right
250  xstart =640:ystart =4400
260  xend =640:yend =1
270  yinc = -yinc
280  GOSUB 380
290 '
300  REM toP left
310  xstart =1:ystart =1
320  xend =1:yend =400
330  xinc = - xinc:yinc = -yinc
340  GOSUB 380
350 '
360  GOTO 80
370'
380  PLOT xstart, ystart
390  DRAW xEND, yEND, inkmask
400  ystart =ystart +yinc
410  xEND =xEND +xinc
420'
```

```
430 IF xstart<1 OR xstart>640 THEN RETU
    RN
440 IF xend <1 OR xend>640 THEN RETURN
450 IF ystart<1 OR ystart>400 THEN RETU
    RN
460 IF yend<1 OR yend>400 THEN RETURN
470'
480 GOSUB 530
490'
500 GOTO 380
510'
520' change colours
530 inkcode=inkcode+1
540 IF inkcode>24 THEN inkcode=0
550 inkmask=inkmask+1
560 IF inkmask>6 THEN inkmask=0
570 IF inkcode=inkmask THEN 550
580 RETURN
```

波纹程序

Amstrad 可以很容易地产生纹波图形。平面就是一个例子：

```
10 REM Interferenec Patterns
20 MODE 1
30 GOSUB 170
40 PRINT CHR$(23); CHR$(scrmode);
50 y%=400

150
```

```
60  FOR x%=640 TO 0 STEP -4
70  ORIGIN 0, 0
80  DRAWR x%, y%, colotr%
90  NEXT
100  GOSUB 170
110  FOR x%=0 TO 640 STEP 4
120  MOVE 640, 0
130  DRAW x%, y%, coloui%
140  NEXT
150  GOSUB 170
160  GOTO 60
170  LOCATE 7, 24:PRINT SPACE$(30);
180  LOCATE 7, 25:PRINT SPACE$(30);
190  LOCATE 7, 24
200  INPUT "Mode 0 -3"; scrmode
210  RESTORE:FOR i=1 TO scrmode
221  READ scrmode$:NEXT
230  LOCATE 20, 24
240  PRINT "     "; scrmode$;
250  LOCATE 7, 25
260  INPUT "Colour 0 -4"; colour%
270  RETURN
280  DATA Normal(forced), XOR, AND, OR
```

非打印控制码

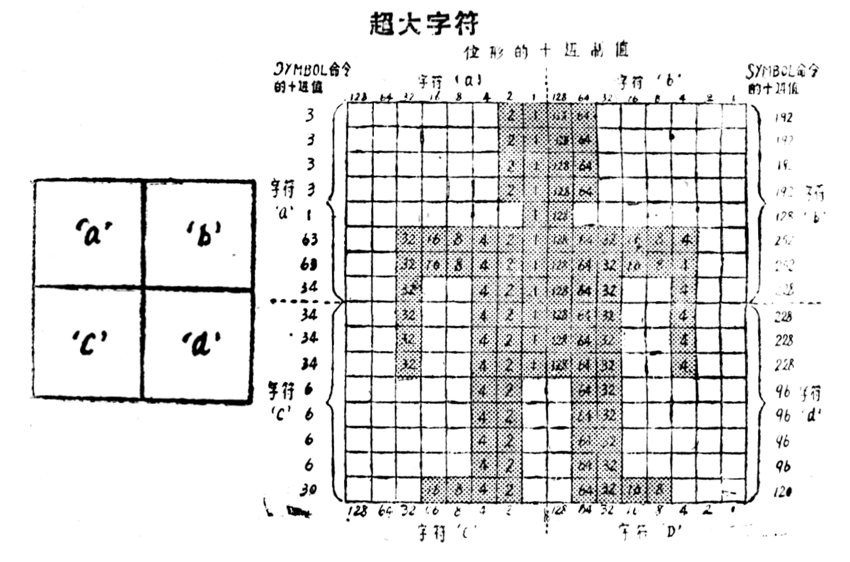
有一组由'PRINT CHR$(n)'产生的屏幕处理命

令，这里n 是0～31之间的值。这些命令不在屏幕上产生字符，除非中间有一个 CHR＄（1），如：'PRINT CHR＄（1）；CHR＄（12）'。这样做可产生32个附加字符，下面的程序显示这32个默认的符号：

```
10  MODE 1
20  FOR char =0 to 31
30  PRINT CHR$（1）；CHR$（char）
40  FOR Panuse =1 TO 100
50  NEXT Pause
60  NEXT char
```

通常，CHR＄（12）清除屏幕，CHR＄（8）为退格。CHR＄（20）清除光标左边的屏幕，CHR＄（18）清除光标下边的屏幕。

超大字符



由于非打印控制码由 CHR＄ 来存取，可以象普通字符一样来处理它们。在 MODE 1和MODE 2中，字符是相当小的，产生大一于般大小的字符是很有用的。一个方法是把一般的字符和控制码连在一起。例如：你可以定义 4 个字符来形成一个方块的象限，我们把它们称为 'a'，'b'，'c'，'d'（见图9.1）。你可以给它们起名为 a＄，b＄，c＄，d＄并把它们连在一起：

现在，每当你'PRINT Sguare＄' 时，这四个字符将在一起出现。这个技术是很简单的：先显示 a＄和 b＄，CHR＄（10）控制换行，把光标移动下一行。CHR＄（8）是一个退格，光标将移到'a'象限下面，然后打印b＄ d＄。下面是个例子：

```
10  MODE 1
20  large.char$ =CHR$(214)+CHR$(215)+
    CHR$(10)+STRING$(2, 8)+CHR$(213)
    +CHR(212)
30  era.char$ =STRING$(2, 32)+CHR$(10)
    +STRING$(2, 8)+STRING$(2, 32)
40  LOCATE 1, 1:GOSUB1000:LOCATE 1, 1:GO
    SUB 2000
50  LOCATE 10, 10:GOSUB 1000:LOCATE 10,
    10:GOSUB 2000
60  GOTO 40
1000 PRINT large.char$;
1010 FOR Paus=1 TO 500:next
1020 RETURN
```

```
2000  PEINT era.charr$; :RETURN
```

为了进一步利用这个技术，我们画一个 16×16 的格子以便在上面设计图形，然后，我们把它分为四个 8 × 8 的方块，你用这些方块可以计算使用 SYMBOL 命令所需的数字。（见图6.1）这些字符方块可以向上面一样连接，也可以按其它方式连接。使用这个方法，可以构成较大的图形。

另一个有用的控制码是 CHR$(31)。这个操作很象命令 LOCATE，所需的两个参数给出光标要移动到位置的列、行坐标。它用于这样的语句：'PRINTCHR$(32); CHR$(S); CHR$(10);'，这个语句把光标移到第五列第十行。这个命令允许产生寄特的字符串和不用 'USING' 命令而打印长数字。请看下面的程序：

```
10  MODE 1:row=1
20  FOR col=1 TO 20
30  LOCATE col, row
40  PRINT "Is this too lo ng? "
50  NEXT
```

运行这个程序后你将发现：当显示的信息太长，而不能在一行显示完时， Amstrad 在显示信息以前打印一个回本和换行，使信息在一行的开始出现。 CHR$(31)允许你避免这一点，它和DEFFN一起使用则产生一个有用的功能：

```
DEF FN Place$(col, row) =CHR$(31)+
CHR$(col)+CHR$(row)
```

现在，你可用 'PRINT FN Place$(13.34)+"Press space"'来代替'100 LOCATE 13.24:PRINT " Press space"'。

弹球子程序

这个子程序可以用于和修改许多弹球游戏。下要我们给出主要过程。

为了在屏幕上移动字符，你必须能控制字位的位置，我们用变量给出这个位置的列行坐标。然后，我们可以在屏幕上的任意位置定位光标和显示字符。为了使字符移动，我们把光标定位在字符的坐标处，并用 CHR $（32）打印一个空格来删除字符。下一步是修改字符的光标，以便重复这个操作。

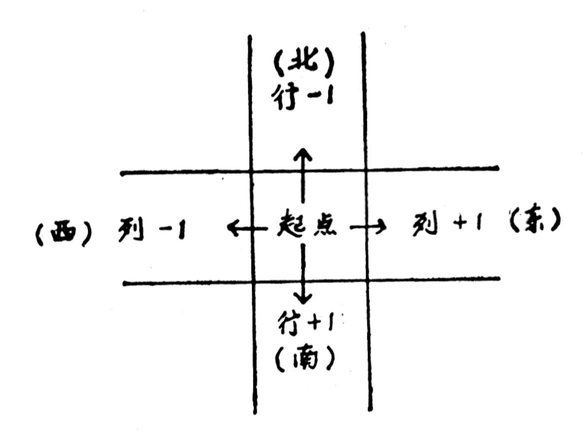
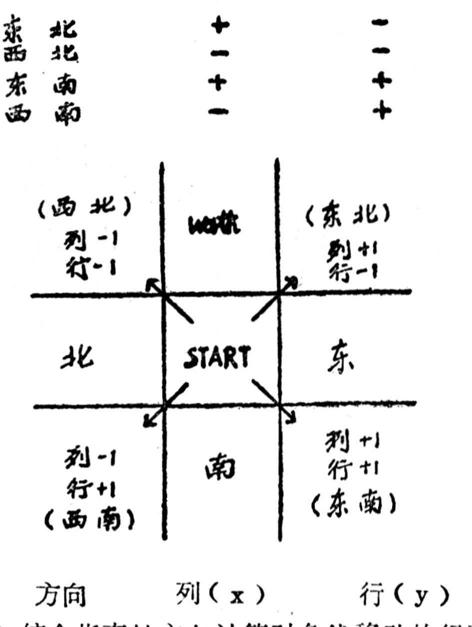


图9.2　按指南针方移用

在屏幕上向上移动字符表示减字符的行座标，向下移动则加字符的行座标。向右移动则相应地加减列 座 标（ 见图9.2 ）。利用同样的原理我们可对西北向，东南向等的移动修改坐标（ 见图9.3和下表）



方向　　　列(x)　　　行(y)

图9.3 结合指南针方向计算对角线移动的行列.

当字符到达屏幕的边沿时，如果我们不改变字符方向，则字符或者从屏幕上消失，或者产生一个出错信息。为了避免这个现象，我们需要测试下一步要移动的坐标。如果超出屏幕极限，则需改变方向。我们使用两个变量`xdir′和`ydir′来控制字符移动的方向，这两个变量的值为 +1或－1 。把它们加到字符的坐标上来修改字符在屏幕上的位置。如果`xdir′为－1，则字符向左移动，因为把`xdir′加到列坐标上，则列坐标减1，同理，`xdir′为＋1，则表示向右移动。`ydir′的作用也相同。下面是这个程序：

```
10  xcoord=320:ycoord=200
15' Directions-south west at the start
20  xdir=-1:ydir=1
30  MODE 1
35' Erase character first
40  MOVE xcoord, ycoord
50  TAG:PRINT CHR$(32);:TAGOFF
55' Update x
60  xcoord=xcoord+xdir
65' Update y
70  ycoord=ycoord+ydir
75' Hit side?
80  IF xcoord<1  OR xcood> 631 THEN THEN
    xdir=-xdri:xcoord=xcoord+xdir
85' Hit top/bottom?
90  IF ycoord<1  OR ycoord>391 THEN ydir=
    -ydir:ycoord=ycoord+ydir
95' Move graphics/text cursors &display
    character
100  MOVE xcoord, ycoord
110  TAG:PRINT "0";:TAGOFF
115 'and repeat
120  GOTO 40
```

窗　口

窗口是屏幕上的长方形区域，你可以把它看做一个小屏

幕。Amsfrad 允允你在屏幕上定义8个这样的窗口。

窗口用 WINDOW 指令来建立，它需要五个参数：窗口号，左右列限，上下行限。对任何窗口，你可使用命令'PRINT #n'，这里n是0～7之间的值。

#0是默认的窗口，当Amstrad 开通时，#0 号窗口被定义为整个屏幕的大小。窗口有时被定义为通道：#8 是是打印机通道，#9是磁带通道。

你定义了几个窗口后，便可用'PRINT #n，在上面显示信息，当窗口被占满时，它在自己的界线内卷动，但卷出的内容，可复盖其它窗口的内容。

用命令'WINDOW SWAPW1，W2' 可以互换窗口，这里W1和W2是要交换的窗口号。

建立窗口

WINDOW 指令的通常形式为'WINDOW #n， 左，右，上，下'，但你可不必按这个顺序给出参数。窗口或通道号是必须先给的，对于'左'和'右'两个值，无论哪个在前，则把较小的一个定为'左'限，'上' '下' 两个值也是如此。

窗口和颜色

象CLS，PRINT，PAPER、PEN 这样的命令，都可以给出窗口号这个参数。例如：PAPER #2，3把2号窗口的背景颜色置成3号墨水的颜色。

下面是在 MODE 0 下使用窗口的简单例子。屏幕分成4个窗口，每个窗口有不同的背景和前景颜色：

```
10' Demonstration of windows
20  MODE 0
```

```
30  WINDOW#1, 1, 10, 1, 12
40  WINDOW #2, 11, 40, 1, 12
50  WINDOW  #3, 1, 10, 12, 25
60  WINDOW #4, 11, 40, 12, 25
70  PAPER #1, 1:PEN#1, 0
80  INK 0, 0' set INK 0 to Black
90  CLS #1
100  PAPER #3, 4:PEN #3, 2
110  CLS#2
120  PAPER #3, 4:PEN #3, 2
130  CLS#3
140  PAPER #4, 7:PEN #4, 13
150  CLS #4
160  FOR count=1 TO 26
170  FOR channel=1 TO 4
180  PRINT # channel, STRING$ (10,  channel
     + count + 64 );
190  NEXT channel, count
200  LOCATE 5, 12:PRINT "THAT' S ALL"
210  GOTO 210
```

使用窗口

用变量来存贮窗口的左/右，上/下限值是一个好做法，此时窗口的定义变为：

```
10  wileft=10:wiright=30
20  witoP=6:wibottom=18
30  WINDOW #1, wileft, wiright, witoP,
```

wibottom

这使得对一个窗口打印非常容易。为了使一个字符串处于打印行的中央，你可使用下面的命令来找到字符串的开始列：

```
10 DEF FN centre(left, right)=ROUND(((
   right-left)/2)-(LEN(a$)/2))
```

窗口复盖

完全分开的窗口易于管理，但是如果你造成了窗口复盖，则会产生一些困难。试运行下面的程序"

现在，因为窗口在10列，12行重叠（复盖），当第一象限卷动时，它卷起了下面象限顶部的一行。右上部的象限也是如此。因此，程序设计时要小心。

轮廓线

本章中的最后一个例子是一个十分复杂的图形程序。给出两个多边形的坐标，一个多边形在另一个之内，我们在每个外面的图形顶点到内图形的最近的顶点间画线，此程序把每个图形分别存贮在两个维数组'x'和'y'中。（n，1)给出外图的角，（n，2）给出内图的点。程序计算哪一个内点离外点最近，并把结果存贮在数组'd'中。下面是程序：

```
10 'Contour Drawing
20 ' Set up your own MODE and colours
30 MODE 2:INK 0, 0:INK 1, 26
40 '
50 ' n1=outer points, n2 inner
60 n1=8:n2=4
70 ' Set up dimensions for outer, inner
```

```
80 ' d() is for differences
90 ' (nearest Points)
100 DIM x(2, n1), y(2, n1), d(n1, n2)
110 '
120 Read in DATA-outer shaPe
130 FOR k=1 TO n1-1
140 READ x(1, k), y(1, k)
150 NEXT
160 ' Move to start of outer shaPe
170 PLOT x(1, 1), y(1, 1)
180 ' Draw lines, Point to Point
190 ' FOR k=1 TO n1-1
200 x1=x(1, k):x2=x(1, k+1)
210 y1=y(1, k):y2=y(1, k+1)
220 a=x2-x1:b=y2-y1
230 DRAWR a, b
240 NEXT
250 ' Save coords of last Point
260 a=XPOS:b=YPOS
270 DRAWR x(1, 1)-a, y(1, 1)-b
280 '
290 ' Read in DATA for inner shaPe
300 FOR k=1 TO n2
310 READ x(2, k)y(2, k)
320 NEXT
330 MOVE x(2, 1), y(2, 1)
```

```
340  ' and draw it
350  FOR k=1 TO n2-1
360  x1=x(2, k):x2=x(2, k+1)
370  y1=y(2, k):y2=y(2, k+1)
380  a=x2-x1:b=y2-y1
390  DRAWR a, b
400  NEXT
410  a=XPOS:b=YPOS
420  DRAWR x(2, 1)-a, y(2, 1)-b
430  '
440  , Now calculate differences
450  ' between outer &inner Points…
460  FOR d=1 TO n1:FOR d2=1 TO n2
470  d(d, d2)=ABS (x(1, d)-x(2, d2))+
    ABS(y(1, d)-y(2, d2))
480  NEXT:NEXT
490  ' …and rank them
500  FOR r=1 TO n1
510  FOR b=1 TO n2
520  bn=0:FOR ɔ=1 TO n2
530  IF d(r, c)>bn THEN bn=d(r, c):cn=c
540  NEXT
550  d(r, cn)=n2-b+1
560  NEXT:NEXT
570  a=x(1, 1):b=y(1, 1):ox=a:oy=b
580  MOVE a, b
```

```
590  'Set detail-high=fine, low=coarse
600  s=50
610  FOR i=1TO s-1
620  FOR f=0 TO i-1:st=st+1/s
630  NEXT f
640  FOR eP =1 TO n1
650  FOR ss=1 TO n2
660  IF d( eP, ss ) =1 THEN j =ss
670  NEXT
680  '
690  'Now draw the' in-betweens'
700  a=x(1, ep) +(st*(x(2,j)-1, ep)))
710  a= ROUND( a )
720  b=y( 1, ed ) + ( st * ( y ( 2, j ) -y ( 1,
     eP)) )
730  b=ROUND( b )
740  DRAWR a-ox, b-oy
750  ox=a:oy=b:NEXT:st=0
760  NEXT
770  ' ===== ALL DONE ======
780  DATA is x, y
790  ' Outer shaPe coorinates
800  DATA 10, 390, 320, 350, 520, 250, 600,
     125, 450, 10, 220, 10, 160, 50, 80, 200
810  ' Inner shaPe coords
820  DATA300, 300, 380, 250, 350, 200, 340, 200
```

第十章　声　音

在本章中，我们将描述产生声音的命令。 声 音 是 由 Amstrad 的内部扩音器产生，但你也可以使用手提式立 体声收录机的耳机来听，这可获得立体声效果。

Basic产生的声音

基本的发声命令是SOUND。‘SOUND1.284’将产生A音符，第一个参数为通道，第二个参数为声音的周期。有三个通道可产生声音，用字母A、B、C来表示。284产生的A音符，频率为440周/秒。

通道和周期

SOUND命令最少有两个参数，但你可用第三个参数定义声音的长度，这个长度是以1/100秒为单位的 。 例 如：‘SOUND 2，284，100’表示在B通道产生1秒钟A 调的‘。第三个参数的最大值为255，大约2.5秒。

表示通道C的参数是4 而不是3 。如果你把立体声系 统连入Amstrad，要注意，左边是通道A，右边是通道 B，中间是通道C。

下面的程序产生存贮各音阶频率的数组：

```
10 REM Music pitches
20,
30 CLS
```

```
40,
50, Data for notes and their values
60, First ( upper )octave
70, DATA F#, F, E, D#, D, C#, C, B,
    A#, A, G#, G
80,
90, Read note names
100 DIM note $(12), note%(9, 12)
110 FOR note%=1 TO 12
120 READ note $ (note%)
130 NEXT
140,
150 PRINT:PRINT "Calculating note values"
160, Calculate note values
170 FOR octave%=4 TO -4 STEP -1
180 realoctave%=octave%+5
190 FOR note%=1 TO 12
200 frequency=440* (2^(octave%+(10-note%)
    /12))
210 Period=ROUND(125000/frequency)
220 PRINT "." ;
230 IF period>4095 THEN 250
240 note%(realoctave%, note%)=period
250,NEXT:NEXT
260,
270 CLS
```

```
280,
290 GOSUB 350
300 CLS
310 GOSUB 510
320 END
330,
340, play notes
350 LOCATE 1, 1:PRINT "Octave" ;
360 LOCATE 1, 2:PRINT "Note"
370 LOCATE 1, 3:PRINT "Period"
380,
390 FOR octave% =5 TO 8
400 FOR note% =12 TO 1 STEP-1
410 LOCATE 7, 1:PRINT octave%
420 LOCATE 6, 2:PRINT note $(note);  "" ;
430 LOCATE 8, 3:PRINT note%(octave%,
    note%);
440 IF note% (cctave%, note) =0 THEN 470
450 SOUND 1, note%(octave%, note%), 100
460 FOR i  =1 TO 1000:NEXT
470 NEXT:NEXT
480 RETURN
490,
500, Scale of C
510 PRINT "Scale of C"
520 FOR octave =5 TO 7
```

```
530 FOR note =12 TO 1 STEP -1
540 IF RIGHT $(note $(note), 1) = "#" THEN
  560
550 SOUND 1, note %(octave, note)
560 NEXT:NEXT
570 RETURN
```

完整的SOUND命令

SOUND命令有 7 个参数：

参数	范围
通道	1 ～128
周期	0～4095
长度	-32768～+32768
开始音量	0～ 7
音量包络	0～15
音调包络	0～15
噪音周期	0～31

并非所有的参数都要使用，最后一个参数可省略，因为它产生一个白噪音。这个参数值的真正范围为0～15， 16～31只是重复0～15的作用。你不必定义音量和音量包络，可用两个引号来省略它们，此时，Amstrad 使用默认值0。

频率

一个纯音由一个正弦波来表示。如果我们把它们画成图10.1的话两个波峰间的距离表示声音的周期，当我们说A 调的频率为440时，就是说每秒钟有440周，一周表示一个完整的正弦波。

音符

音乐的音符由字母A～G表示，有些可加#和b 符号 。一个音符和下一个音符之间的音程叫做半音，两个半音为一个全音。

大多数音乐是以音阶为基础的，所遵循的顺序为全音，全音、半音、全音、 音、全音、半音，我们把这个顺序简写为TTSTTTS。使用这个方法，我们可以写一个子程序，来从数组中的任意音符开始音奏音阶。首先我们建立一个串数组存贮‘TTSTTTS’序列，再建一个整数组存贮音 调的序号：

```
10 Scale $ = "TTSTTTS"
20 DIM Scale%(7)
```

然后，我们使用布尔逻辑给‘scale% ’赋值：

```
30 FOR note% =1 TO 7
40 astep $ =MID $ (scale $, note%, 1)
50 scale%(note%) = -2* (astep $ = "T" ) -
   ( astep $ = "S" )
60 NEXT
```

下面与上有同样作用：

```
30 FOR note% =1 TO 7
40 IF MID $(scale $, note%, 1) = "T" THEN
   scale%(note%) = 2
50 IF MID $(scale $, note%, 1) = "S" THEN
   scale%(note%) =1
60 NEXT
```

如果你已经建立了数据，则可使用下面的子程序演奏音

阶：

```
100 INPUT "octave" , oct
110 INPUT "note" , anote
120 INPUT "duration" , dur
130 REM Trap illegal values…
140 SOUND 1, note%(oct, anote), dur
150 FOR count =1 TO 7
160 anote =anote - scale %(count)
170 IF anote<1 THEN oct =oct - 1
180 IF anote =0 THEN anote =12
190 IF anote = -1 THEN anote =11
200 SOUND 1, note%(oct, anote), dur
210 NEXT
```

和弦

有许多种和弦，从大调到小调及7度和音。这里，我们只能最简单的介绍。

主和弦由同时奏3个音组成。Amstrad 有3个通道，可产生和弦。首先你必须决定从哪个音调开始，然后计算组成主和弦的音。C主和弦从C音阶得到，使用C、E、G音：

```
SOUND 1, 119, 100
SOUND 2, 95, 100
SOUND 4, 80, 100
```

小调和弦需要第2个音降半音，C小调为C、E^b，G：

```
SOUND 1, 119, 100
SOUND 2, 100, 100
SOUND 4, 80, 100
```

音调包络线

ENT命令是关于音调包络的，它允许你定义当一个音演奏时如何改变其频率。

Amstrad 不能象乐器一样平滑地改变音的高度，你只能阶梯式地改变。如果你要把一个音的频率升高100 单位，你必须先决定这个音有多长，假设长1秒。这样，我们可分成100步，每步长 0.01秒，步值为－1，因为周期越小，声音越高。这些值在ENT指令中如下：

```
1000 ENT 1, 100, -1, 1
```

第一个数为包络号，第二个数是步数，第三个数每步的改变值，第四个数是每步的时间。

对于ENT命令，只有15个包络号可定义，包络0为默认值，并且不能被改变。步数的范围为0～239，步值在 －182～127之间，步长为0～255。

音调包络可以是负数。假如负号的音调包络在音符奏完前结束时，则这个音调包络重复，直至这个音符奏完。下面的例子说明了重复和非重复音调包络间的差别：

```
10 ENT -1, 100, 5, 1
20 PRINT "Negative envelope"
30 SOUND 1, 284, 400, 7, 0, 1
40 INPUt "Press ENTER to continue"
50 INPUT a $
60 ENT 1, 100, 5, 1
70 PRINT "Positive envelope"
80 SOUND 1, 284, 400, 7, 0, 1
```

设计一个音调包络

对于每个音调包络数，可定义5个变音部分。设计一个音调包络的最好方法是在坐标纸上画出阶梯式的图形。你把图分成5部分，每部分定义三个值：步数、步值、步长（见图10.5）。

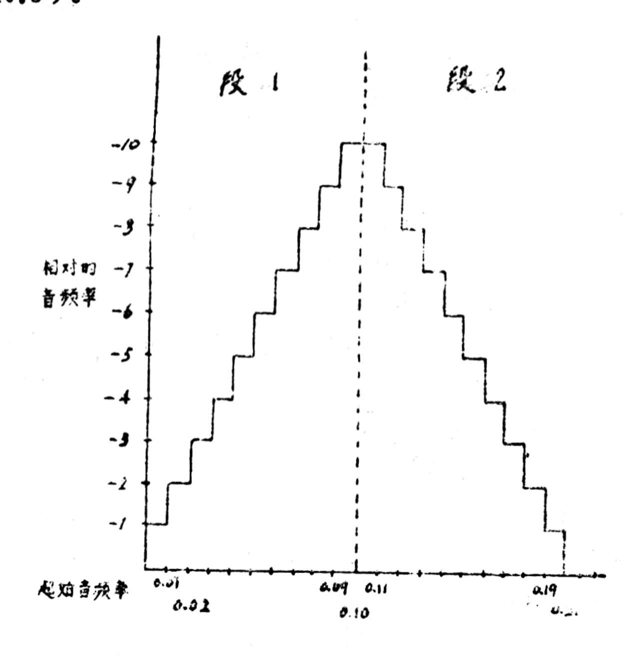


图10.5

时间（秒）

图10.5 在音状态中使用2个段，第一段通过降低频率来升高音阶，第二段到音阶降到起始音频率，这是由命令“ENT 1, 10, -1, 1, 10, 1, 1”来定义的。

音量包络

音量包络定义的方法差不多与音调包络相同。

音量状态允许你改变音量或扩大声音。

实验音量包络

熟习音量状态的最好方法是实验。定义音量状态的命令是ENV及一些参数，请看下面的例子。

```
10 ENV generator
20 note=300
30, Define volume as high & low
40 hivol=15:lovol=8
50 vol=hivol-lovol
60, Define number of steps for each section
70 asteps=3
80 dsteps=3
90 ssteps=4
100 rsteps=4
110, Calculate volume increases for each
     phase
120 ainc=ROUND(hivol/asteps)
130 dinc=-ROUND(vol/dsteps)
140 sinc=0
150 rinc=-ROUND(lovol/rsteps)
160' Define ADSR dursation
170 adur=2
180 ddur=1
```

```
190 sdur =10
200 rdur =1
210 MODE 2
220 LOCATE 1, 3:PRINT "ATTACK:"
230 LOCATE 1, 4:PRINT "DECAY:"
240 LOCATE 1, 5:PRINT "SUSTAIN:"
250 LOCATE 1, 6:PRINT "RELEASE: "
260 LOCATE 10,1:PRINT "Steps" , "Change"
    "Duration" , "Total Change" , "Length "
270 LOCATE 12, 3:PRINT asteps, ainc, adur,
    asteps*ainc, asteps*adur
280 LOCATE 12, 4:PRINT dsteps, dinc, ddur,
    dsteps*dinc, dsteps*ddur
290 LOCATE 12, 5:PRINT ssteps, sinc, sdur,
    ssteps*sinc, ssteps*sdur
300 LOCATE 12, 6:PRINT rsteps, rinc, rdur ,
    rsteps*rinc, rsteps*rdur
310 '
320 'Define ENV 1
330 ENV 1, asteps, ainc, adur, dsteps, dinc,
    ddur, ssteps, sinc, sdur , rsteps , rinc ,
    rdur
340 LOCATE 1,8:PRINT "Envelope is:channel
 "; asteps; ", "; ainc; " ,"adur"; dsteps; " ,
 "; dinc; " , "; ddur; " , " ; ssteps ; " ,
 "; sinc; " , " sdur ; " , "; rsteps ; " ,
```

```
"; rinc;", ";rdur"
350 'Calculate note length
360 length =asteps*adur + dsteps*ddur + ssteps *
sdur + rsteps*rdur
370 LOCATE 1, 10:PRINT "Total length is" ;
length
380 fv =asteps*ainc + dsteps*dinc + steps*sinc +
rsteps*rinc
390 PRINT "Final volume is" ; fv
400 If fv <0 THEN length =length - fv
410 'play note
420 SOUND 1, note, length, 0, 1
430 END
```

噪音周期

这是SOUND命令的最后一个参数，并且是可选的。这个参数定义噪声的频率，其值为正，，表示把噪音加到声音上。试运行下面的程序：

```
10 FOR n =TO 31
20 RRINT n
30 SOUND 1, 200, 100, , , n
40 NEXT
```

通道、聚集和抑制

通道参数不仅定义哪个通道产生声音，也用于确保音在同时发出以及抑制这个音等。通道参数所允许的值及意义如

下：

1	使用通道A
2	使用通道B
4	使用通道C
8	同A汇合
16	同B汇合
32	同C汇合
64	保持
128	释放

要使用这个表，首先要决定如何使用声音，然后在SOUND命令中写入适当的的参数值。例如，我们有两个音符并希望同时演奏（汇合），这两个音符一个通过A通道，另一个通过B通道。这样，一个通道值为 17 ，由16加1 而得，1表示通道A，16表示同通道B汇合。同理，另一个通道值将为10。

参数值为64表示保持声音，即把声音放入一个队列中，当这个声音到达队头时，就停在那里，直到对那个通道发了一个RELEASE命令。

通道参数的第 7 位（十进制组 128）表示把队列中声音移到队头，并使之马上演奏。

声音队列

每个声音通道有一个等候奏出的声音队列。SOUND命令把一个音的参数放入所定义通道的队列中。一个队列中可有四个声音。Amstrad 有一些处理声音队列的特殊命令。SQ用1，2，4作为参数（表示通道A，B，C），并且回送一个表示通道状态的数。如果SQ的参数不是1，2或4，则产

生出错信息'lmproper argument'（不正确的参数）。下面为SQ回送值的意义（由一个8位字节表示）：

位	意义
0，1，2	队列中的空位数
3	同A汇合在队头
4	同B汇合在队头
5	同C汇合在队头
6	保持在队头
7	通道有效

为了利用SQ回送字节中的信息，你必须执行 AND 操作。例如，为了得出A通道队列中有多少空闲位置，你必须把这个字节和7做AND操作。

```
100 channelA.stat = SQ（1）
110 free.entries = channelA.stat AND 7
```

为了检查在队头的声音是否和另一个通道汇合，则要把回送的字节同56做AND操作。如果结果不是零，则表示有一个汇合。为了得知同哪个通道汇合，你要用32(通道C)，16（通道B）或8（通道A）和结果做AND操作；其结果就是所聚集的通道：

```
100 channel.Astat = SQ(1)
110 rvous = channel.Astat AND 56
120 rvousA = rvous AND 8
130 rvousB = rvous AND 16
140 rvousC = rvous AND 32
150 IF rvousA = 8 THEN…
160 IF rvousB = 16 THEN…
```

```
170 IF rvousc=32 THEN…
180 'etc
```

为了测试在队头的声音是否被置成保持，用64ANDSQ的结果，得出的值为64，则表示有声音在队头被保持，需要用RELEAS命令使它奏出。

用128 AND SQ的结果，则得出通道是否在206作用，即是否正在奏一个声音。

RELEASE

这个命令用于释放在给定通道内的保持的音符。对这个命令你可使用1～7之间的参数。RELEASE 1释放A的声音队列，RELEASE 1释放所有的通道队列，RELEASE 6释放B和C中的队列。通过保持和释放，可控制音的顺序和同步。

ON SQ…GOSUB

ONSQ的用法非常象'EVERY'中断命令（见第十二章）。它检查什么时候通道声音队列的空闲空间变为可用的，你必须用1，2或4做为表示通道的参数。当发现一个空位时，则程序转到GOSUB指出的行号，使你可以在这个队列中放一个新的音。注意SQ和SOUND禁止ONSQ。

声调变音

当两个相关的频率一起奏出时，则产生这个现象。为了计算两个频率共鸣的值，首先以赫艺（Hz）来选音高，然后计算其和及差的绝对值。下面要在Amstrad手册的频率表中查找最相近的周期数，并把其做为参数加到SOUND命令中。

下面的程序说明如何计算：

```
10 REM Ring modulation
20 DEF FN freq(note)=440*(2^(octave+(10-
note)/12))
30 DEF FN period(freq)=ROUND(125000/freq)
40 FOR count=1 TO 10
50 octave=INT(RND(1)*2)-1
60 note1=INT(RND(1)*12)+1
70 note2=INT(RND(1)*12)+1
80 IF note1=note2 THEN 60
90 freq1=FNfreq(note1)
100 freq2=FNfreq(note2)
110 Sumfreq=freq1+freq2
120 diffreq=ABS(freq1-freq2)
130 Period1=FNperiod(sumfreq)
140 period2=FNperiod(diffreq)
150 IF period1<3822 OR period2<3822 THEN
60
160 IF period1>12 OR period2>12 THEN 60
170 SOUND 10, period1, 100, 7
180 SOUND 17, period2, 100, 7
190 NEXT
```

第十一章　磁带系统

Amstrad 的内部磁带机是很可靠和方便的，你可以很好的利用它。

装载程序

你在你的新机器上第一件想做的事是运行一个个商用程序，为此，你必须把此程序从磁带中装入。你键入 RUN“ ”，然后按(ENTER)，Amstrad 将提示你按磁带机上的 PLAY 键和按另一个键。当你做完这些操作后，Amstrad 将转动磁带马达，把程序从磁带读入存贮器，然后运行它。装入程序的快速方法是按CTRL和数字盘上的小ENTER键。LOAD”将把磁带上的下一个程序装入存贮器，但不运行这个程序。RUN和LOAD后面可以跟一个带引号的程序名。

存贮程序

为了存贮程序，在确保不复盖其它程序的情况下，把磁带放入磁带机，然后键入SAVE“程序名”。计算机将提示你按磁带机上的PLAY和RECORD键，然后计算机将把这个程序存入磁带中。程序名最多可有16个字符，可使用数字。如果你使用的字符超过16个，则第17以后的字符被丢掉。当文件存贮时，把所有的文件名都变成了大写。

快速写

用SPEED WRITE1 命令可使磁带记录信息的速度增加一倍，这对于存贮一个较长的程序是很有用的。用SPEED WRITE 0 可恢复正常的速度。

存贮可选操作

当存贮程序时，有若干可选操作供你使用。例如，你在程序名前加了一个感叹号，象SAVE“! CIRCLES”，则不显示应有的提示。注意，当程序记录时，! 被删除。当装载信息时如果你想使用自己的提示，则可在程序名前加!，使Amstrad 不显示正常提示而显示你的提示。

保护程序：P操作

有时，你不想让别人看你的程序，你可在SAVE命令中加，P，如：SAVE “game”，P。此时，只能使用RUN或CHAIN 指令来把程序装入存贮器，不能只是装入，然后再列程序清单。当你使用P存贮一个程序时，你必须确保这个程序已不需要修改了，因为包括你自己在内，谁也不能列出这个程序。如果你使用ESC来中断一个加保护的程序。则程序将消失。如果加保护的程序有格式错误，当程序由于错误而停止时，Amstrad 的保护系统将把程序从存贮器中删除。

文件

所有存在磁带上的信息，无论是程序、字符串数组屏幕图象，以及机器码程序都称为文件。

第四章中有一个简单的数据库程序，它利用磁带机存贮一个字符串数组的数据。以这种方法产生的文件称为“顺序文本文件”。由于存贮介质的关系，信息在磁带上必须以线性方式存贮并且译成ASCII码形式。顺序文件的存贮和恢复是很慢的。你不能重新记录文件的一部分，如果你想得到文件尾的信息，你必须处理整个文件。随机存取的文件的效率较高，但它们只能建立在快速磁带或磁盘上。

ASCII格式：，A操作

Amstrad 用数字表示所有的Basic命令。从地址438开始存贮程序。下面一段程序表示，这个程序怎样存入RAM：

```
10 REM line 10
20 Basic program revealer
30 MODE 2
40 address=438
45 PRINT "Address Line"
46 PRINT "       Bytes"
50 bytes=PEEK(address)
60 line.no=PEEK(address)+2
70 PRINT address; bytes; line.no;
80 FOR count =bytes to bytes-2
90 conts =PEEK(address)+count
100 IF conts>31 AND conts<128 THEN PRINT
CHR $(conts); ELSE PRINT conts;
110 NEXT
120 PRINT
130 address =address+bytes
```

140 GOTO 50

/A操作把程序用ASCII码格式存入磁带。这样存入的程序较长，因为ASCII码占用较多的空间，但这样的程序装载较可靠并且可做为正常文本文件处理。

记录（存贮）数据

为了打开磁带上的文件，你可使用命令 ‘OPENOUT “文件名” ’。为了把数据记录到磁带上，你要使用命令 ‘PRINT#9 ’。PRINT9 #后面必须有一个逗号，这个命令的用法与屏幕打印完全一样。当你向磁带送完数据时，你必须用CLOSEOUT命令关闭这个文件，因为Amstrad要在文件的结尾记录一个‘文件尾 ’标志，以便查找时识别。

由于你可在PRINT#中使用变量，在把数据存入磁带前，你可以在屏幕上检测文件处理程序。这样可允许你较快较容易地调试你的程序。其做法是：在你编制程序的开始为PRINT通道设一个变量，如：device = 0 。然后，使用 PRINT #derice 在任意磁带上写程序，当你满意这个子程序为工作时，把 device的值改为9，则把数字送入了磁带。

在把信息送入磁带前，Amstrad 先把信息写入缓存，仅当缓存写满时或发出CLOSEOUT 命令，才把信息送入磁带机。

WRITE

这个命令不常用，它很象PRINT命令。它后面可有一个表示输出对象的表达式，如：WRITE#9，WRITE#0。再后面是要写的项目，如：WRITE#0 “This” ，Value ，

That $。用WRITE命令输出时，数字用逗号隔开，字符串用引号括上。‘WRITE#9，value1,astring $,value2 ’相当于下面的PRINT语句：

```
PRINT #9, value1
PRINT #9, astring $
PRINT #9, value2
```

使用WRITE命令允许你用INPUT命令读入语句。

检索数据

在你从磁带口读入文件以前，必须打开这个文件 。为此，我们使用命令‘OPENIN“文件名”。企图 OPENIN一个程序则产生出错信息‘File type error ’(文件类型错)，除非这个程序已用,A操作存贮了。你也不能运行一个ASCII码文件，除非它是一个Basic程序。

在一个数据文件中，用前几个字节来描述文件是一个好做法。在第 4 章的数据库程序中，在把数据送入数组之前送入行的项数。当打开这个文件时，先读入行的项数，并把它作为读数组的循环计算上限。

EOF

EOF 是“End OF File”(文件结尾)的缩写，当你不知道文件多长时，可使用它。这个功能经常这样使用：‘IF EOF THEN GOTO XXXX ’，例如：

```
10 OPENIN""
20 IF EOF THEN 60
30 INPUT #9, a $
40 PRINT a $
```

```
50 GOTO 20
60 CLOSEIN
```

这个程序从一个文件中读所有的数据，并在屏幕上显示，当遇到EOF时，表示文件读完，则关闭文件并且结束程序。

LINE INPUT

如果你存贮含有逗号的一串数据，逗号将象PRINT#9中的分隔符那样处理。字符串‘Mary Jones，191 The Arenue，Durham ’将做为三个分开的项来处理。试运行下面的程序：

```
10 CLS:OPENOUT "TEST"
20 PRINT #9, "abc, def, ghi"
30 CLOSEOUT
```

现在，卷回磁带，运行下面的数据检索程序：

```
10 CLS:OPENIN "TEST"
20 IF EOF THEN 70
30 INPUT #9, a$
40 count = count + 1
50 PRINT count, a$
60 GOTO 20
70 CLOSEIN
```

这个程序说明了 INPUT 如何处理逗号，和数据如何被分开。如果你把30行改为‘LINE INPUT# 9，a $’，再重新运行这个程序，则可看到：LINE INPUT 避免了这个问题。

存贮主存中的数据块

SAVE 命令有若干可选操作。例如：你可存贮在屏幕上产生的图象。这个命令如下：

SAVE "picture", B, &C000, 16384, &C000

B表示要存贮的主存数据块，&C000 是RAM中的视频存贮器的起始地址，16384 是要存贮的主存块长度（所有的屏幕方式使用16K RAM），最后一个参数表示重新装载这个信息的起始地址。下面的程序画一个干扰图形并把它存入磁带：

```
10 MODE 2
20 y =400
30 FOR x =639 TO 0 STEP -3
40 ORIGIN 0, 0
50 DRAWR x, y
60 NEXT
70 FOR x =0 TO 639 STEP 3
80 MOVE 639, 0
90 DRAW x, y
100 NEXT
110 SPEEDWRITE 1
120 SAVE "! interference", B, &C000, 16384
&C000
```

这个程序运行完后，键入 NEW 并卷回磁带，再键入下面的程序：

```
10 MODE 0
```

20 LOAD "! interference"

当你运行这个程序时，你将看到图形不是从头到尾画出，显示的行顺序为0，8，16等。这是由于你把 MODE 2产生的图形在MODEO显示的结果。注意：当读入数据时，无颜色闪烁，但在谈存贮块之间和文件被装入后，有许多颜色闪烁。

了解如何在屏幕上显示图象，可使你用旧图形得到新效果，甚至可使屏幕上下两半互换显示等。你只需知道屏幕起始地址&C000，和200行的每一行占有80 个字节。根据屏幕的映象方法，很容易PEEK每个字节并把内容存入磁带。

当然，显示的所有信息，包括磁带操作的提示也将记录下来，所以在 SAVE 和 LOAD 命令中的文件名前，不要忘记使用感叹号。

存贮字符

如果你重新设计了一些或全部 Amstrad 的字符，你也许想在其它程序中使用它们，但是对每个程序都键入这些重新定义字符的命令是很麻烦的。然而，由于字符定义存贮在 RAM 中，你可把这些字符存入磁带并把它装入其它程序。

字符在 RAM 中从地址&A500开始存贮，每个字符由 8 个字节定义。有224个打印字符（32～255），所以结束地址为42240＋（224＊8），即44031，共1792字节长。为了存贮字符集合，你要使用'SAVE "CHAR.SET"，B，42240，1792，42240'。

如果你对128～170的ASCII 码字符重新定义，你首先要计算起始地址（42240＋（128－32）＊8），然后要计算

存贮字节的长度，此时为344个字节。存贮命令如下：

SAVE "chars", B, 43008, 344, 43008

存贮机器码

存贮机器码子程序可节省时间，因为你不必在每个使用它的程序的开始键入建立这个子程序的DATA 语句，你只需写这样一个子程序即可：

```
10 REM Create and Save machine code
20 DATA 205, 96, 187, 50, 23, 171, 201
30 MEMORY 43798
40 FOR count=1 TO 7
50 READ value
60 POKE 43799+count, value
70 NEXT
80 SAVE "PEEK.TXTSCRN", B, 43799, 8,
   43799
```

编目文件

当你键入 CAT 时，Amstrad 将显示和键入LOAD同样的提示。当你再按 PLAY 和一个键后，计算机将读磁带上的每个文件，然后显示每个文件的名子，并用符号标注文件类型：

$ -Normal Basic Program 正常的BASIC程序

% -Protected Basic Program 保护的BASIC程序

* -ASCII file ASCII码文件

& - Binary file 二进制文件

如果你想停止编目过程，你必须多按几次 ESC键。因为 Amstrad 不允许你在读或写磁带时中断。

每个文件有一个头，它含有文件名和文件类型。头后面是文件，文件分成许多数据块，每块有自己的头，用来说明数据块号。

链接程序

你可用存在 RAM 中的程序来 RUN 第二个程序。但是，当你这样做时，你便丢掉了现有的程序，因为新程序将原来的程序复盖了。Amstrad 提供了一个连接功能，允许存贮器中的程序再连接一个磁带上的程序并能使第二个程序使用第一个程序的数据。

CHAIN

在一个程序中可使用 CHAIN 来装载和运行另一个磁带上的程序，而不丢失第一个程序建立的变量值。如果一个程序很长，则可以分成两部分连续运行。CHAIN 的用法如下：

```
10000 CHAIN "NEXT.PROG"
```

你也可规定从新程序的那个行号开始运行：

```
CHAIN "PAPT.TWO", 3000
```

这个命令把名为PAPT.TWO的程序装入，并从3000行开始运行。

CHAIN MERGE（链接合并）

这是一个较复杂的 CHAIN 命令形式。它允许你把磁带上的一个程序和存贮器中的一个程序合并。存贮器中的程

序中若有和磁带程序相同的行号，则被删除，这些行被新装入的行复盖。在这个命令中你可加入‘DELETE’和行号范围，例如：

5000 CHAIN MERGE “PART.THREE”，9000，DELETE

3000—6000

这个语句把名为PART.THREE 的程序和当前存贮器中的程序合并，并从9000行开始执行合并后的程序。存贮器程序的3000～6000行被DELETE命令删除。下面是两个较复杂的例子。

1000 CHAIN MERGE“PART.FOUR”，DELETE—1000

2000 CHAIN MERGE “PART.FIVE”，6000，DELETE 9000—

使用 CHAIN 命令必须小心。所有用 DEFFN 定义的功能都不再存在，因此新合成的程序要重新定义它们。任何 ON ERROR GOTO 条件和 FOR…NEXT，WHILE…WEND或GOSOB 结构都不再存在。DATA 语句被恢复，所有打开的磁带数据文件不再存在。DEFSTR，DEFINT和DEFREAL 指令也不执行。

ROM调用

有若干 ROM 子程序供你的程序调用，以使你得到一些有用的功能。例如，CALL &BC6B 可使提示信息显示或不显示，如果要不显示提示，A 寄存且必须为0，否则，显示提示。因此，你必须使用一个子程序来把相应的值装入

就是A寄存器，然后调用 ROM 子程序：

```
10 MEMORY 43879:address=43879
20 POKE 43880, 62:REM LD A, n
30 POKE 43881, 255:REM data
40 POKE 43882, 205:REM CALL
50 POKE 43883, 107:REM low byte of &BC6B
60 POKE 43884, 188:REM high byte of &BC6B
70 POKE 43885, 201:REM RET- 'return '
```

磁带出错信息

出错信息可分成两部分：读错误和写错误。

读错误不经常发生，Amstrad 的磁带系统是较可靠的。有三种类型的读错误：a，b和c。‘Read error b’表示读数据不正确，要倒回磁带重新读。Read error a’和‘Read error c’表示较大的错误。

避免磁带错误的最简单方法是仔细选择磁带。要使用高质量的磁带，不要长于90分钟。对待磁带也要仔细，不要太经常地重复使用。

当 Amstrad 正在写磁带时，不要中断它。

如果你试图装载一个在另一个机器上用 SPEED WRITE 1 记录的程序或数据，则可产生读错误。如果你想复制你的程序，应用 SPEED WRITE 0 来写。

提示‘Rewind taPe’（倒带）表示数据块的顺序乱了。

仅有一个写出错信息，它不经常出现，因为仅当磁带系统不能把信息足够快地写到磁带上时，才会产生这个错误。

第十二章 中 断

Amstrad 的Basic 有一组其它家族微机所没有的命令。这些命令允许你有在特殊的间隔执行的子程序并为程序员提供了许多方便。例如：在屏幕上显示一个实时时钟，在程序运行时演奏音乐或在定期的间隔中显示提示。做这些事情的方法依赖于“中断”这个概念。

定时器

Amstrad 中有5种定时器。TIME 命令是其中的一种，但它与中断无关。你可利用的定时器的编号是0～3，它们以1/50秒为单位，从0记到255。

中断是为要处理的子程序而由某个计时器发出的对系统的请求。当有一个中断请求时，系统将暂停当前的工作来为这个请求服务。在 Basic 中，这意味着程序可在FOR……NEXT 循环中被中断，然后执行产生中断的子程序，完后，继续执行这个循环。这个过程非常象 GOSUB…RETURN，计算机记住中断的断点，当子程序执行完时，便返回那点。

EVERY

最有用的中断处理命令是 EVERY。EVERY 必须有两个参数。一个 GOSUB 和一个行号。例如：‘EVERY 50，1 GOSUB 1000’，这个命令的意思为：在1号定时器每记50个数以后；从1000行开始执行子程序。第一个数是间隔，表示中断产生的频度。第二个数是所使用的定时器

号。由 EVERY 产生的中断只能和 GOSUB 组合，不能和 GOTO 组合。子程序必须以 RETURN 结尾，以便返回断点。

下面是一个例子：

```
10 MODE 1
20 EVERY 50, 1 GOSUB 40
30 GOTO 30
35 REM End of main routine
40 n=n+1
50 LOCATE 10, 10
60 PRINT n; "seconds";
70 RETURN
```

这个例子给出了一个简单的秒记时器；每秒钟执行一次把变量n加1并显示变量值的子程序。

下面的例子说明如何在一个程序内改变中断的定时：

```
10 MODE 1
20 n=1
30 EVERY n, 3 GOSUB 1000
40 GOTO 40
1000 PRINT n
1010 n=n+1
1020 IF n>20 THEN n=1
1030 EVERY n, 3 GOSUB 1000
1040 RETURN
```

关闭和开放中断

下面的例子说明了在 Basic 中使用中断的一些问 题。这个程序产生三个围着显示器跳的数，每个数代表用于产生中断的计时器并且每个数有自己的‘跳子程序 ’:

```
10 'Bouncing numbers
20 CLS
30 X0 =10:y0 =10
40 X1 =X0:y1 =y0
50 X2 =X0:y2 =y0
60 b0$ = "0" :b1 $ = "1" :b2$ = "2"
70 dX0 =1:dy0 =1
80 dX1 = -1:dy1 =1
90 dX2 =1:dy2 = -1
100 '
110 EVERY 4, 0 GOSUB 170
120 EVERY 3, 1 GOSUB 270
130 EVERY 5, 2 GOSUB 370
140 GOTO 140
150 '
160 'Number 0
170 DI:LOCATE ×0, y0
180 PRINT "  ";
190 X0 =X0 + d×0
200 y0 =y0 + dy0
210 IF x0 > 39 OR x0 < 2 THEN dx0 = -dx0
```

```
220 IF y0>22 OR y0 <2 THEN dy0 = -dy0
230 LOCATE x0, y0, PRINT b0$;
240 EI:RETURN
250 '
260 'Number 1
270 DI: LOCATE X1, Y1
280 PRINT "  ";
290 x1 =x1+dx1
300 y1 =y1+dy1
310 IF x1>39 OR x1 <2 THEN dx1 = -dx1
320 IF y1>22 OR y1 <2 THEN dy1 = -dy1
330 LOCATE x1, y1:PRINT b1$;
340 EI:RETURN
350 '
360 'Number 2
370 DI:LOCATE x2, y2:PRINT "  "
380 x2 =x2+dx2
390 y2 =y2+dy2
400 IF x2 >39 OR x2 <2 THEN dx2 = -dx2
410 IF y2 >22 OR y2 <2 THEN dy2 = -dy2
420 LOCATE x2, y2:PRINT b2$;
430 EI:RETURN
```

这个程序中使用了两个重要的中断命令：DI和EI。DI为关闭中断，当你想使一段程序不被中断时，使用这个命令并且把它做为这段程序的第一条指令。EI 为开放中断，用于把关闭中断状态恢复为允许中断状态。

定时器优先权

由于4个计时器都可发中断请求，它们之间的互相影响可产生一些问题，因此，系统要登记这些请求，并决定先响应哪一个。事实上，有一个中断请求队列，所有发出的请求，都要加入这个队列中。对于每个请求有不同的优先权，3号计时器有最高的优先权。如果三个计时器的中断请求同时到达，则最先响应3号计时器的请求。0号计时器有最低的优先权，其次是1号、2号计时器。

优先权系统可产生一些问题。例如：如果把跳数子程序中1号或2号计数器的间隔速率设为较低的值，则没有时间去响应0号计数器的中断请求了。

可以用ESC键表演排队。按ESC产生一个必要中断，程序暂停，直到按了另一个键。如果第二次也按了ESC键，则程序不可再执行，控制返回到直接方式。但是，时间定数器不象这个中断，因此中断请求可一直发出，即使程序已明显的停止了。

试运行下面的程序，你将看到，1号计时器以1秒为间隔记数，2号计时器以两秒为间隔计数。按ESC键暂停这个程序，几秒钟以后再按空格键，则两个计时器同步较好。如果你使程序暂停几分钟，则只接收2号计时器的中断：

```
10 CLS
20 EVERY 50, 1 GOSUB 1000
30 EVERY 100, 2 GOSUB 2000
40 GOTO 40
1000 count1 = count1 + 1
1010 LOCATE 1, 1:PRINT"Timer 1" ; count1;
```

```
1020 RETURN
2000 count2 = count2 + 2
2010 LOCATE 1, 2:PRINT "Timer 2"; count2
2020 RETURN
```

当使用 EVERY 时，你必须仔细检查中断驱动子程序要处理多长时间，如果所花的时间大于中断的间隔，则可能产生问题。

有两个利用计时器显示时间的方法：从某一点计时或做为一个时钟。后者需要用户在程序中键入某一点的时间，下面是一个计时子程序：

```
10 count = TIME
20 CLS
30 EVERY 50, 3 GOSUB 10000
40 GOTO 40
50 '
9999 'Clock subroutine
10000 counts = TIME - const
10010 seconds = ROUND(counts/300)
10020 minutes = ROUND(seconds/60)
10030 hours = ROUND(seconds/3600)
10040 seconds = seconds MOD 60
10050 minutes = minutes MOD 60
10060 hours = hours MOD 24
10070 second$ = STR$(seconds)
10080 minute$ = STR$(minutes)
10090 hours$ = STR$(hours)
```

```
10100 second$ =RIGHT(second$, 2)
10110 minute$ =RIGHT$(second$, 2)
10120 hour$ =RIGHT$(hour$, 2)
10130 LOCATE 1, 1
10140 PRINT hour$; ":", minute$; ":";
second$;
10150 RETURN
```

下面是一个数字时钟子程序：

```
10 DEF FN strip$(anyvar)=RIGHT$(STR $
(anyvar), 2)
20 CLS
30 INPUT "Hours = ", hrs
40 INPUT "Minutes =", mins
50 INPUT "Seconds =", secs
60 EVERY 50, 3 GOSUB 10010
70 CLS
80 GOTO 80
90 '
10000 'Time Update
10010 DI:secs=secs+1
10020 IF secs>59 THEN secs=0:mins=mins+
10030 IF mins>59 THEN mins=0:hrs = hrs+1
10040 IF hrs>23 THEN hrs=0
10050 LOCATE 1, 1
10060 PRINT FN strip$(hrs); ":";
10070 PRINT FN strip$(mins); ":";
```

```
10080 PRINT FN strip$(secs);
10090 EI:RETURN
```

AFTER

AFTER 是另一个中断命令，但它不如 EVERY 有用。AFTER 指示在指定定时器计了一定的数后执行某一子程序。AFTER 是一次性命令，当遇到这个命令时，相应的定时器置 0，与这个定时器有关的任何其它中断都被取消，当到达所规定的间隔时，则执行指定的子程序。AFTER 和EVERY都专用指定的定时器，EVERY 也象 AFTER 一样，取消对一个定时器的其它中断命令。下面的例子说明怎样使 AFTER 象 EVERY 一样起作用：

```
10 REM AFTER Demo
20 CLS
30 REM Subroutines which reset
40 'themselves
50 AFTER 50, 1 GOSUB 1000
60 AFTER 100, 2 GOSUB 2000
70 GOTO 70
80 '
1000 PRINT "Subroutine 1"
1010 AFTER 50, 1 GOSUB 1000
1020 RETURN
1030 '
2000 PRINT TAB(5); "Subroutine 2"
2010 AFTER 100, 2 GOSUB 2000
```

```
2020 RETURN
```

ON BREAK GOSUB

象上面说明的一样，ESC 键象一个高优先权中断。按两次 ESC 键程序将停止并使控制回到直接方式。但是ON BREAK GOSUB 允许你在用户试图中断程序时转到你自己的子程序。你可使用这个指令来保护你的程序，不被任何人停止，列出或改变。它也可做为瞬时“帮助”工具，可在程序中的任意处访问之。下面是“不可中断”程序的基本原理：

```
10 ON BREAK GOSUB 10000
15 ON ERROR GOTO 10000
20 REM Rest of Program
30 REM
9000 GOTO 20
10000 RETURN
```

为了在屏幕上显示‘帮助’，你需要做的是：当按两次 ESC 键时，把控制转移到子程序的“帮助”部分。为此，你必须知道中断发生在什么地方，因此你必须在一个变量中保存操作的轨迹，例如：

```
10 ON BREAK GOSUB 10000
20 REM Main Program
30 REM Menu for operations selection
1000 REM End of main section
3000 REM Invoice section
3010 Place$ = “invoice”
```

```
3020 REM Rest of invoicing routine
3990 RETURN
3999 REM End of invoices
10000 REM HELP
10010 IF Place$ = "invoice" THEN GOSUB 1100
10020 IF Place$ = "receipt" THEN GOSUB 1200
10030 REM Rest of Help screens
10040 RETURN
11000 MODE 1:LOCATE 1, 1:PRINT "Help on
Invoicing"
11010 REM Rest of help info
11090 RETURN
```

为了研究程序，你可使用 ON BREAK 跳到显示变量值的子程序：

```
10 ON BREAK GOSUB 10000
20 REM Rest of Program
9999 END
10000 PRINT "Length of string="; LEN
(words$)
10010 PRINT "count="; count
10020 PRINT "Press space to continue"
10025 akey$ = " "
10030 WHILE akey$ <> CHR$(32)
10040 akey$ =INKEY$
10050 WEND
10060 RETURN
```

同样，你可使用 ON BREAK 退出一个麻烦的状态。例如：你用命令 SPEED KEY 1，1 把键设成了快速重复状态，你将发现：如果你按两次 ESC 键中断程序，则键盘变得不好用了。键重复得太快以至不可能键入命令来恢复正常。为了使键的重复值置为默认值，我们把 ESC，ESC 移到一个子程序，这个子程序经过 ROM 调用重新设置重复值并且没有 RETURN，这样程序便不再执行并返回到直接方式：

```
10 ON BREAK GOSUB 10000
20 SPEED KEY 1, 1
30 REM Rest of Program
9999 REM
10000 CALL &BB00
```

你也可使机器关掉而结束程序，即使 CTRL SHITF ESC 不能重新引导。为此可使用 POKE 48622，201，而 POKE 48622，195 使机器能重新引导。ESC 可用CALL 47947 使之无效。

ON ERROR GOTO

ON ERROR 非常象 ON BREAK，如果产生错误，此命令使程序跳到一个给定的行号。在上面的例子中，你也可增加指令‘15 ON ERROR GOTO 10000，这样使产生格式错时，把键盘处理复位为正常。

RESUME

ON ERROR GOTO有一个配对的命令，就是RESUM。

RESUM 后面可跟一个行号或 NEXT。如果你在程序中先使用了一个 ON ERROR GOTO，当产生错误时，程序将跳到GOTO 后的行号。在这个行号处，你可设一段错误处理操作并在结尾放一个 RESUME NEXT 命令。这样可使控制转回到产生错误语句的后一个语句。你也可把控制转到一规定的行号，如：RESUME 5500。

ERR和ERL

ERR 回送错误号。ERL 回送出现错误的行号。ERR 和ERL 是系统变量，也是数字变量，因此你可写特定的错误处理程序来改正错误而不使程序停止。例如，你可报告错误类型和错误码，而不必停止程序：

```
10 ON ERROR GOTO 10000
20 REM Rest Program
9999 END
10000 PRINT "ERROR"; ERR; "In line":
ERL
10010 RESUME NEXT
```

如果你正在从磁带文件中谈内容并且遇到一个‘EOF met’错误，则下面的程序可解决问题：

```
10000 IF ERR=24 THEN CLOSE:PRINT
 "Unexpected end of file in line"; ERL
10010 CLOSE:RESUME NEXT
```

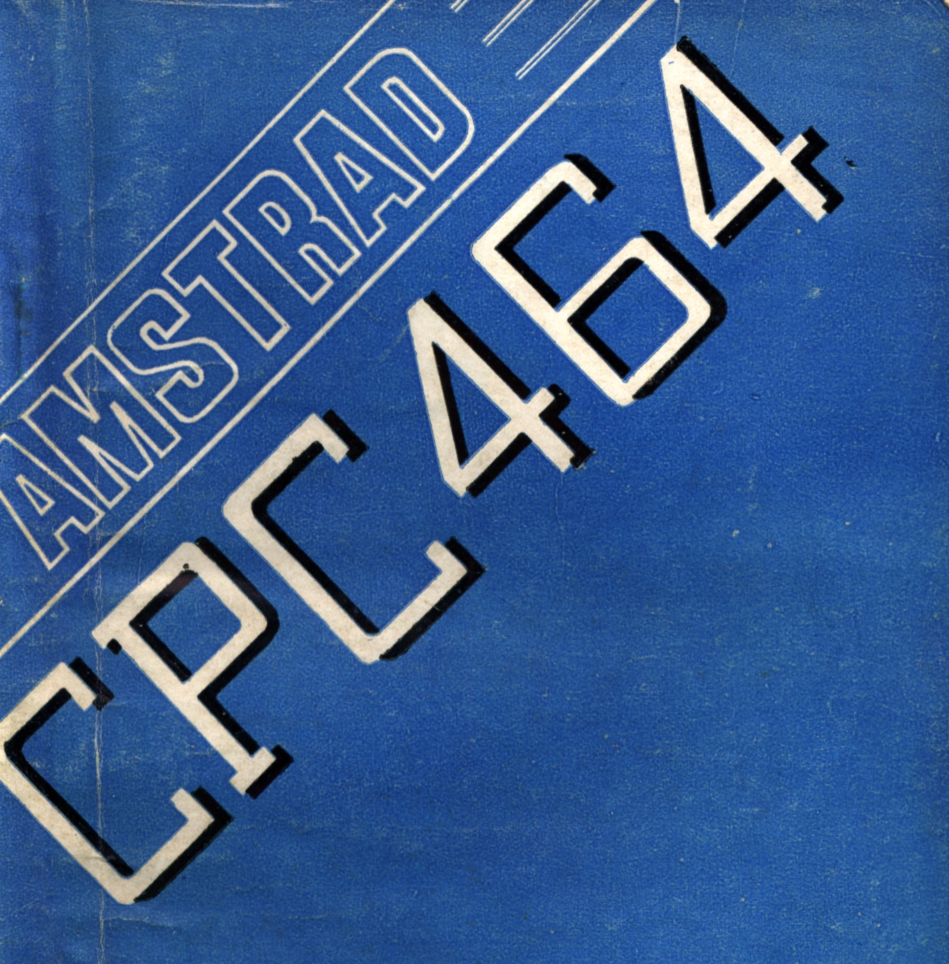
你也可使用 ON 来把控制转移到错误处理子程序的行号，如：ON ERR GOSUB 1000, 2002, 2500.

REMALIN

REMAIN 用于回送定时器中的计数并且关闭定时器中断请求。它的用法如 'dummy＝REMAIN(n)'。括号中的数必须是整数并且是4个计时器之一。REMAIN 回送给定时器的时间值并把这个定时器置为 0。你可使用这个命令来关闭来自一个定时器的中断。你也可用这个命令测试一个定时器计数到何处，然后对那个定时器复位中断间隔或把一个新值赋给那个计时器。如果一个定时器是不可用的(即没对那个计时器进行中断分配)，则 REMAIN 回送0。

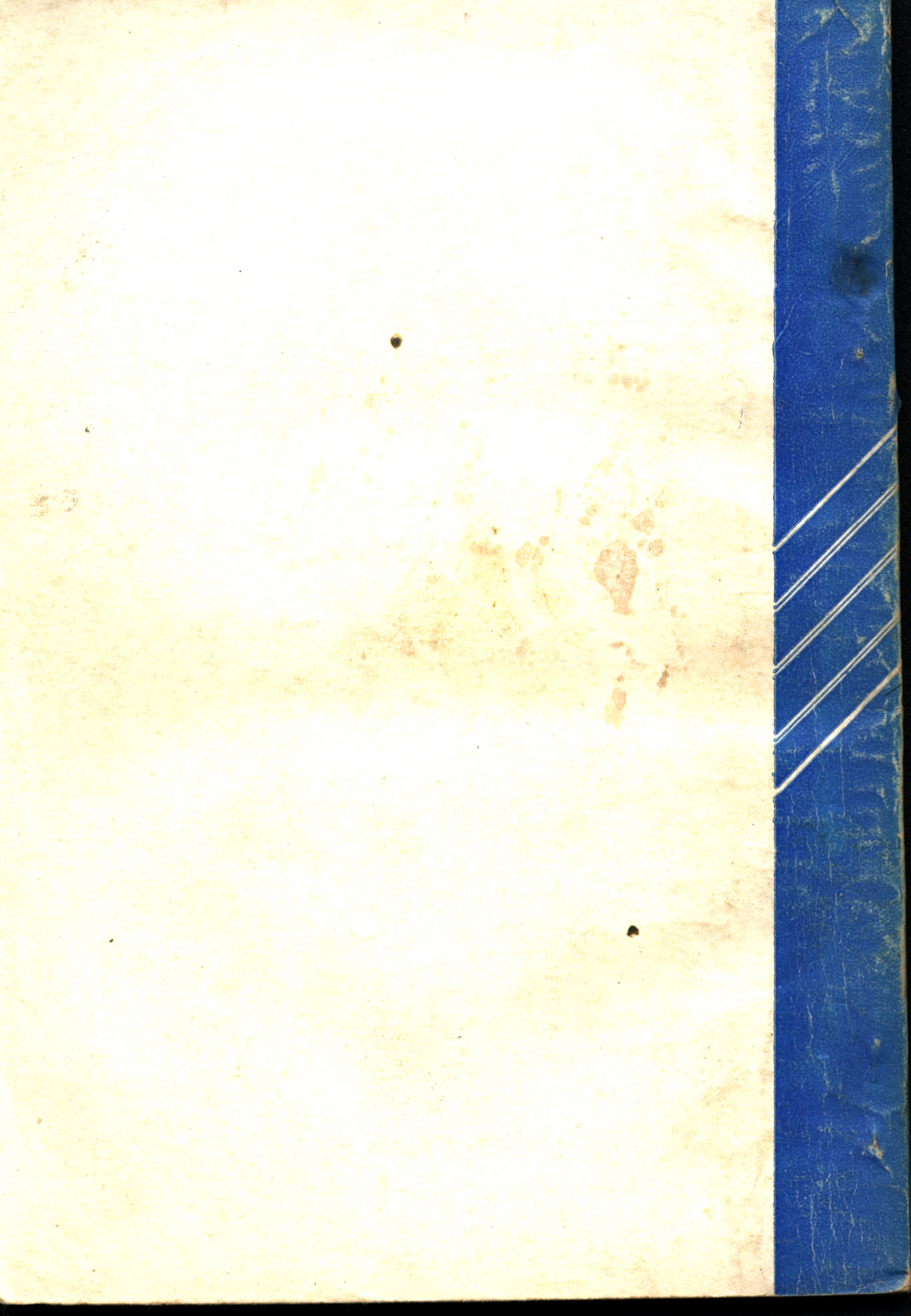






程序员手册

南方信息企业有限公司







程序员手册

南方信息企业有限公司


AMSTRAD
CPC
MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA
ACME
https://acpc.me/